Python 的内置函数 __import__

说明：导入模块。

`__import__` 是 Python 的一个底层内置函数，通常不建议直接使用，而是由 `import` 语句在内部调用。但在某些动态导入场景下（如通过字符串决定导入模块名时），可以使用它。

函数说明

```
def __import__(name, globals=None, locals=None, fromlist=(), level=0):
    '''
    说明：导入模块。

    :param name: 模块名
    :param globals: 全局命名空间字典
    :param locals: 局部命名空间字典
    :param fromlist: 要导入的名称列表
    :param level: 绝对导入（0）还是相对导入
    :return: 导入的模块
    '''
```

说明

导入模块。

参数

- `name` - 模块名
- `globals` - 全局命名空间字典
- `locals` - 局部命名空间字典
- `fromlist` - 要导入的名称列表
- `level` - 绝对导入（0）还是相对导入

返回值

导入的模块。

示例

```
# 相当于 import math
math = __import__('math')
```

```
print(math)
print(math.pi)

# 相当于 from os import path
os_path = __import__('os', fromlist=['path'])
print(os_path.path)
```

推荐阅读

Python 官方文档

Python 的内置函数 abs

说明：计算绝对值。

函数说明

```
def abs(x):
    '''
    计算参数的绝对值

    :param x: 要计算的值
    :return: x 的绝对值
    '''
```

说明

通过调用对象的 `__abs__` 方法，计算对象绝对值。

- `int` 和 `float` 返回绝对值
- `complex` 返回复数的模
- 自定义类型需要实现 `__abs__` 方法

参数

- `x` - 要计算的值

返回值

`x` 的绝对值。

示例

```
print(abs(-100))
print(abs(-233.3333))
print(abs(0))
print(abs(128))
print(abs(-3 + 4j))          # 复数的模

# 自定义类型
class Point:
    def __init__(self, x, y):
        self.x, self.y = x, y
```

```
    def __abs__(self):
        return (self.x ** 2 + self.y ** 2) ** 0.5

p = Point(3, 4)
print(abs(p))  # 5.0
```

推荐阅读

- Python 官方文档
- 内置函数 complex

Python 的内置函数 aiter

说明：获取异步可迭代对象的迭代器。

函数说明

```
def aiter(async_iterable):
    '''
    获取异步可迭代对象的迭代器

    :param async_iterable: 一个异步可迭代对象
    :return: 参数的异步迭代器
    '''
```

说明

通过调用对象的 `__aiter__` 方法，获取异步可迭代对象的迭代器。

可以参考同步版本的 iter 函数。 注意， `aiter` 没有两个参数的版本，这一点与 iter 函数不同,。

参数

- `async_iterable` - 一个异步可迭代对象

返回值

`async_iterable` 的异步迭代器。

示例

```
import asyncio

# 异步迭代器
class AsyncIterator:
    def __init__(self, stop):
        self.__stop = stop
        self.__current = 0

    async def __anext__(self):
        if self.__current < self.__stop:
            await asyncio.sleep(0.1)  # 模拟异步操作
            self.__current += 1
            return self.__current - 1
        else:
```

```
            raise StopAsyncIteration

# 异步可迭代对象
class AsyncIterable:
    def __init__(self, stop):
        self.__iterator = AsyncIterator(stop)

    def __aiter__(self):
        return self.__iterator

# 创建异步可迭代对象
async_iterable = AsyncIterable(10)

# 获取迭代器
async_iterator = aiter(async_iterable)
print(async_iterator)
```

推荐阅读

- Python 官方文档
- 迭代器和可迭代对象
- 内置函数 anext
- 内置函数 iter
- 内置函数 next

Python 的内置函数 all

说明：判断可迭代对象内容是否全部为真值。

函数说明

```
def all(iterable):
    '''
    判断可迭代对象内容是否全部为真值

    :param iterable: 一个可迭代对象
    :return: 如果 iterable 的所有元素均为真值则返回 True，否则返回 False；如果 iterable 是空的，返回 True
    '''
```

说明

判断可迭代对象的元素是否 **全部** 为真值，空的可迭代对象也视为元素全部为真值。

真值与假值的判断规则请参考 bool 函数的 说明。

参数

- `iterable` - 一个可迭代对象

返回值

- 如果 `iterable` 的所有元素均为真值，则返回 `True`
- 如果 `iterable` 中存在假值，否则返回 `False`
- 如果 `iterable` 是空的，返回 `True`

示例

```
print(all([]))              # 空的可迭代对象返回 True

print(all(range(10)))       # 0...9
print(all(range(1, 10)))    # 1...9
```

推荐阅读

- Python 官方文档

- 内置函数 any

Python 的内置函数 anext

说明：获取异步迭代器的下一数据项。

函数说明

```
async def anext(async_iterator):
    '''
    获取异步迭代器的下一数据项，
    没有下一项时产生 StopAsyncIteration 异常

    :param async_iterator: 一个异步迭代器
    :return: 迭代器的下一项
    '''
```

说明

通过调用对象的 `__anext__` 方法，获取异步迭代器的下一数据项。

异步迭代器没有下一项时，应当从 `__anext__` 方法中通过抛出 `StopAsyncIteration` 异常进行通知，`anext` 函数会将该异常继续抛出。

可以参考同步版本的 next 函数。

参数

- `async_iterator` - 一个异步迭代器

返回值

迭代器的下一项数据的 `awaitable` 对象；如果迭代器没有下一项，则抛出 `StopAsyncIteration` 异常。

```
async def anext(async_iterator, default):
    '''
    获取异步迭代器的下一数据项，没有下一项时返回 default

    :param async_iterator: 一个异步迭代器
    :param default: 默认值
    :return: 迭代器的下一项
    '''
```

说明

通过调用对象的 `__anext__` 方法，获取异步迭代器的下一数据项。

异步迭代器没有下一项时，应当从 `__anext__` 方法中通过抛出 `StopAsyncIteration` 异常进行通知，`anext` 函数会返回 `default` 参数传入的默认值。

可以参考同步版本的 next 函数。

参数

- `async_iterator` - 一个异步迭代器
- `default` - 迭代器没有下一项时返回的默认值

返回值

迭代器的下一项数据的 `awaitable` 对象；如果迭代器没有下一项，则返回 `default` 参数传入的默认值。

示例

```
import asyncio

# 异步迭代器类
class AsyncIterator:
    def __init__(self, stop):
        self.__stop = stop
        self.__current = 0

    async def __anext__(self):
        if self.__current < self.__stop:
            await asyncio.sleep(0.1)  # 模拟异步操作
            self.__current += 1
            return self.__current - 1
        else:
            raise StopAsyncIteration

# 异步函数
async def main():
    # 异步迭代器对象
    async_iterator = AsyncIterator(10)

    # 迭代
    while (value := await anext(async_iterator, None)) is not None:
        print(value)

# 启动
asyncio.run(main())
```

推荐阅读

- Python 官方文档
- 迭代器和可迭代对象
- 内置函数 aiter
- 内置函数 iter
- 内置函数 next

Python 的内置函数 any

说明：判断可迭代对象内容是否存在真值。

函数说明

```
def any(iterable):
    '''
    判断可迭代对象内容是否存在真值

    :param iterable: 一个可迭代对象
    :return: 如果 iterable 中存在真值则返回 True，否则返回 False；如果 iterable 是空的，返回 False
    '''
```

说明

判断可迭代对象内容是否 **存在** 真值，空的可迭代对象不存在真值。

真值与假值的判断规则请参考 bool 函数的 说明。

参数

- `iterable` - 一个可迭代对象

返回值

- 如果 `iterable` 存在真值，则返回 `True`
- 如果 `iterable` 全部是假值，否则返回 `False`
- 如果 `iterable` 是空的，返回 `False`

示例

```
print(any([]))                      # 空的可迭代对象返回 False
print(any(range(10)))               # 0...9
print(any([0 for _ in range(0)]))   # 0...0
```

推荐阅读

- Python 官方文档

- 内置函数 all

Python 的内置函数 ascii

说明：将对象转换为 ASCII 字符串。

函数说明

```
def ascii(obj):
    '''
    转换为字符串（调用对象的 `__repr__` 方法），非 ASCII 字符将被转义

    :param obj: 一个对象
    :return: obj.__repr__ 返回的字符串
    '''
```

说明

通过调用对象的 `__repr__` 方法，将其转换为字符串，字符串中的非 ASCII 字符将被转义。

如果不希望转义，可以使用 repr 函数。

参数

- `obj` - 要转换的对象

返回值

转换后的 ASCII 字符串。

示例

```
class Cat:
    def __repr__(self) -> str:
        return "I am a 喵喵"

cat = Cat()

print(ascii(cat))
print(repr(cat))
```

推荐阅读

- Python 官方文档

- 内置函数 repr

Python 的内置函数 bin

说明：将一个整数转换为带前缀 `0b` 的二进制字符串。

函数说明

```
def bin(x:int):
    '''
    将一个整数转换为带前缀 `0b` 的二进制字符串

    :param x: 一个整数
    :return: x 的二进制字符串形式
    '''
```

说明

将一个整数转换为带 `0b` 前缀的二进制字符串。结果是一个合法的 Python 表达式。

如果参数 `x` 不是整数，则它必须定义 `__index__` 方法，该函数返回一个整数；`bin` 会将其返回的整数转换为带 `0b` 前缀的二进制字符串。

参数

- `x` - 要转换的整数

返回值

以 `0b` 开头的二进制字符串。

示例

```
print('0 =>', bin(0))           # 0b0
print('10 =>', bin(10))         # 0b1010
print('1024 =>', bin(1024))     # 0b10000000000

class Custom:
    def __index__(self) -> int:
        return 1024

obj = Custom()
print('obj =>', bin(obj))
```

推荐阅读

- Python 官方文档
- 内置函数 oct
- 内置函数 hex

Python 的内置函数 bool

说明：类型转换为 `bool`。

函数说明

```
class bool(x=False):
    '''
    类型转换为 bool

    :param x: 一个变量
    :return: 转换为 bool 后的值
    '''
```

说明

将变量转换为 `bool` 类型。

以下值会被转换为 `False`：

- `None`
- `False` 自身
- 值为 0 的数值类型变量，包括但不限于 `int`，`float`
- 长度为 0 的容器，包括但不限于空字符串、空元组、空列表
- `__bool__` 方法返回 `False` 的对象
- `__len__` 方法返回 0 的对象

除此之外的值都将被转换为 `True`。

参数

- `x` - 要转换的变量，默认为 `False`

返回值

转换为 `bool` 类型后的值

示例

```
print("0 是", bool(0))
print("10 是", bool(10))
print("'hello world' 是", bool('hello world'))
```

```
print("'' 是", bool(''))
print("None 是", bool(None))
```

推荐阅读

- Python 官方文档

Python 的内置函数 breakpoint

说明：设置断点。

函数说明

```
def breakpoint():
    '''
    调用位置进入调试器
    '''
```

说明

用于设置断点，在调用位置会进入调试器。当程序执行到 `breakpoint()` 时，会调用 `sys.breakpointhook()`。

默认情况下会启动 `pdb` 调试器（Python 标准库中的调试器）。

可以通过设置 `PYTHONBREAKPOINT` 环境变量来改变调试器行为：

- `PYTHONBREAKPOINT=0` 禁用所有断点
- `PYTHONBREAKPOINT=pudb.set_trace` 使用 PUDB 调试器

参数

无

返回值

无

示例

```
print('hello')
breakpoint()
print('world')
```

调试器的命令

执行控制

- `q(uit)` - 退出调试器并终止程序
- `c(ontinue)` - 继续执行直到遇到下一个断点
- `n(ext)` - 执行当前行，停在下一行（不进入函数）
- `s(tep)` - 执行当前行，进入被调用的函数
- `r(eturn)` - 继续执行直到当前函数返回
- `unt(il)` [行号] - 继续执行直到到达指定行号或当前函数返回

断点管理

- `b(reak)` [文件名:]行号 - 在指定行设置断点
- `b(reak)` 函数名 - 在函数入口设置断点
- `cl(ear)` [中断编号] - 清除断点（不指定参数则清除所有）
- `tbreak` [参数列表] - 设置临时断点（命中一次后自动删除）
- `disable` [中断编号] - 禁用断点
- `enable` [中断编号] - 启用断点
- `ignore` 中断编号 [次数] - 忽略断点指定次数

代码导航

- `l(ist)` [first[, last]] - 列出当前行周围的代码
- `ll` - 列出当前函数或框架的所有代码
- `w(here)` - 显示当前执行堆栈
- `u(p)` - 向上移动一个堆栈帧
- `d(own)` - 向下移动一个堆栈帧
- `a(rgs)` - 显示当前函数的参数

变量检查

- `p` 表达式 - 打印表达式的值
- `pp` 表达式 - 漂亮打印表达式的值
- `whatis` 参数 - 打印参数的类型
- `display` [表达式] - 每次停止时自动显示表达式值
- `undisplay` [表达式] - 停止显示表达式值

代码执行

- `!` 语句 - 执行 Python 语句（可省略感叹号）
- `run` [参数列表] - 重启调试的程序（仅在 pdb 启动后可用）
- `restart` [参数列表] - 同 run

变量操作

- `interact` - 启动交互式解释器
- `debug` code - 进入递归调试器调试代码

调试器配置

- `source` 文件名 - 从文件执行 pdb 命令
- `commands` [中断编号] - 设置断点命中时要执行的命令
- `j(ump)` 行号 - 跳转到指定行号（改变执行流）
- `retval` - 打印函数返回的值

其他

- `h(elp)` - 显示帮助信息
- `alias` [别名 [命令]] - 创建命令别名
- `unalias` 别名 - 删除别名
- `condition` 中断编号 表达式 - 设置条件断点
- `disable` [中断编号] - 禁用断点
- `enable` [中断编号] - 启用断点
- `longlist` | `ll` - 列出当前函数的所有源代码

推荐阅读

- Python 官方文档
- The Python Debugger

Python 的内置函数 bytearray

说明：类型转换为 `bytearray`。

`bytearray` 是可变的字节数组，拥有可变序列的大部分常见方法，请参考 list。

`bytearray` 也拥有不可变字节数组 bytes 的大部分常见方法。

函数说明

```
class bytearray(x=b''):
    '''
    创建 bytearray

    :param x: 要转换的变量
    :return: x 转换为 bytearray 后的值
    '''
```

说明

将变量转换为 `bytearray` 类型。

参数

- `x` - 要转换为 `bytearray` 的变量，默认为 `b''`，即创建一个空的 `bytearray`
  - 类型为 `bytes` 时拷贝该内容创建一个 `bytearray`
  - 类型为 `int` 时创建一个长度为 `x` 元素值全部为 0 的 `bytearray`

返回值

转换为 `bytearray` 后的值。

```
class bytearray(x, encoding):
    '''
    创建 bytearray

    :param x: 要转换的变量
    :param encoding: 编码
    :return: x 转换为 bytearray 后的值
    '''
```

说明

将变量转换为 `bytearray` 类型。

参数

- `x` - 要转换为 `bytearray` 的变量，通常是 `str`
- `encoding` - 转换时使用的编码

返回值

转换为 `bytearray` 后的值。

```
class bytearray(x, encoding, errors):
    '''
    创建 bytearray

    :param x: 要转换的变量
    :param encoding: 编码
    :param errors: 编码错误的处理方式
    :return: x 转换为 bytearray 后的值
    '''
```

说明

将变量转换为 `bytearray` 类型。

参数

- `x` - 要转换为 `bytearray` 的变量，通常是 `str`
- `encoding` - 转换时使用的编码
- `errors` - 编码错误的处理方式

返回值

转换为 `bytearray` 后的值。

编码错误的处理方式

- `None` 或 `'strict'` - 编码错误时产生 `ValueError` 异常
- `'ignore'` - 忽略错误，通常会丢失数据
- `'replace'` - 使用 `'?'` 替换有错误的位置
- `'surrogateescape'` - 使用 `U+DC08` 至 `U+DCFF` 之间的值表示不正确的字节
- `'xmlcharrefreplace'` - 编码格式不支持的字符将被替换为相应的 XML 字符引用，仅在写入时可用

- `'backslashreplace'` - 使用反斜杠转义表示错误的数据
- `'namereplace'` - 使用 `\N{...}` 转义表示错误的数据

示例

```
print(bytearray(10))                            # 长度为 10，元素全部为全 0
print(bytearray(b'hello'))                      # from bytes
print(bytearray('hello', encoding='utf-8'))     # from str
```

推荐阅读

- Python 官方文档
- 内置函数 bytes
- 内置函数 list
- 内置函数 tuple

Python 的内置函数 bytes

说明：类型转换为 bytes。

`bytes` 是不可变的字节数组；与之相对，bytearray 是可变的字节数组。

函数说明

```
class bytes(x=b''):
    '''
    创建 bytes

    :param x: 要转换的变量
    :return: x 转换为 bytes 后的值
    '''
```

说明

将变量转换为 `bytes` 类型。

参数

- `x` - 要转换为 `bytes` 的变量，默认为 `b''`，即创建一个空的 `bytes`
  - 类型为 `bytes` 时拷贝该内容创建一个新的 `bytes`
  - 类型为 `int` 时创建一个长度为 `x` 元素值全部为 0 的 `bytes`

返回值

转换为 `bytes` 后的值。

```
class bytes(x, encoding):
    '''
    创建 bytes

    :param x: 要转换的变量
    :param encoding: 编码
    :return: x 转换为 bytes 后的值
    '''
```

说明

将变量转换为 `bytes` 类型。

参数

- `x` - 要转换为 `bytes` 的变量，通常是 `str`
- `encoding` - 转换时使用的编码

返回值

转换为 `bytes` 后的值。

```
class bytes(x, encoding, errors):
    '''
    创建 bytes

    :param x: 要转换的变量
    :param encoding: 编码
    :param errors: 编码错误的处理方式
    :return: x 转换为 bytes 后的值
    '''
```

说明

将变量转换为 `bytes` 类型。

参数

- `x` - 要转换为 `bytes` 的变量，通常是 `str`
- `encoding` - 转换时使用的编码
- `errors` - 编码错误的处理方式

返回值

转换为 `bytes` 后的值。

编码错误的处理方式

- `None` 或 `'strict'` - 编码错误时产生 `ValueError` 异常
- `'ignore'` - 忽略错误，通常会丢失数据
- `'replace'` - 使用 `'?'` 替换有错误的位置
- `'surrogateescape'` - 使用 `U+DC08` 至 `U+DCFF` 之间的值表示不正确的字节
- `'xmlcharrefreplace'` - 编码格式不支持的字符将被替换为相应的 XML 字符引用，仅在写入时可用
- `'backslashreplace'` - 使用反斜杠转义表示错误的数据

- `'namereplace'` - 使用 `\N{...}` 转义表示错误的数据

示例

```
print(b'abcd')                                  # 拷贝该另一个 bytes
print(bytes(10))                                # 长度为 10，元素全部为 0
print(bytes('hello', encoding='utf-8'))         # 将字符串编码为 bytes
```

推荐阅读

- Python 官方文档
- 内置函数 bytearray
- 内置函数 list
- 内置函数 tuple

Python 的内置函数 callable

说明：判断对象是否可调用。

函数说明

```
def callable(obj):
    '''
    判断对象是否可调用

    :param obj: 一个对象
    :return: 如果 obj 可以调用则返回 True，否则返回 False
    '''
```

说明

判断对象是否可以被调用，自定义类通过实现 `__call__` 方法实现调用过程。

请注意，即使 `callable` 判断对象可以被调用，在实际调用对象时仍然可能会失败；但如果 `callable` 判断对象不可被调用，那么调用对象必定失败。

类是可调用的，调用类将返回一个新的实例，也即构造函数。

参数

- `obj` - 要判断的变量

返回值

- 如果对象可以被调用，则返回 `True`
- 如果对象不可被调用，则返回 `False`

示例

```
class Foo:
    pass

class Bar:
    def __call__(self):
        pass

def func():
    pass
```

```
print('数值', callable(10))
print('lambda', callable(lambda x:x))
print('函数', callable(func))
print('Foo 对象', callable(Foo()))
print('Bar 对象', callable(Bar()))
```

可调用对象的类型

1. **函数**：包括内置函数、自定义函数和 lambda 表达式。

```
def greet():
    print("Hello!")
print(callable(greet))  # 输出: True
```

2. **方法**：类中定义的方法。

```
class MyClass:
    def method(self):
        pass
obj = MyClass()
print(callable(obj.method))  # 输出: True
```

3. **类**：类本身是可调用的，因为调用类会创建一个实例。

```
class MyClass:
    pass
print(callable(MyClass))  # 输出: True
```

4. **实现了 `__call__` 方法的对象**：如果一个类实现了 `__call__` 方法，其实例也可以被调用。

```
class CallableClass:
    def __call__(self):
        print("Called!")
obj = CallableClass()
print(callable(obj))  # 输出: True
```

推荐阅读

- Python 官方文档

Python 的内置函数 chr

说明：单个 Unicode 字符的整数编码转字符串。

函数说明

```
def chr(x:int):
    '''
    单个 Unicode 字符的整数编码转字符串，
    实参的合法范围是 0 到 0x10FFFF，不在范围内会产生 ValueError 异常

    :param x: 单个 Unicode 字符的的编码
    :return: 单个 Unicode 字符的字符串
    '''
```

说明

将单个 Unicode 字符的整数编码转换为字符串。

单个 Unicode 字符的整数编码有效范围是 0 到 0x10FFFF，如果参数不在这个范围内，将会抛出 `ValueError` 异常。

反向的转换请使用 ord 函数。

参数

- `x` - 单个 Unicode 字符的整数编码
  - 有效范围是 0 到 0x10FFFF
  - 如果参数不在这个范围内，将会抛出 `ValueError` 异常。

返回值

`x` 编码的字符串；如果 `x` 不在有效范围内，则抛出 `ValueError` 异常。

示例

```
# ASCII 兼容字符
print('65 编码的字符是', chr(65))              # 'A'
print('97 编码的字符是', chr(97))              # 'a'

# 汉字
print('30002 编码的字符是', chr(30002))        # '甲'
print('20013 编码的字符是', chr(20013))        # '中'
```

```python
# 特殊符号
print('8364 编码的字符是', chr(8364))          # '€'
print('9992 编码的字符是', chr(9992))          # '✈'

# emoji 表情
print('128512 编码的字符是', chr(128512))      # '😀'
print('129322 编码的字符是', chr(129322))      # '🤪'
```

推荐阅读

- Python 官方文档
- 内置函数 ord

Python 的内置函数 classmethod

说明：把一个方法封装成类方法。

函数说明

```
def classmethod(fn):
    '''
    把一个方法封装成类方法

    :param fn: 要封装的方法
    :return: 封装后的方法
    '''
```

说明

将一个方法封装成类方法，通常以 装饰器 的形式使用。

类方法隐含的第一个参数就是类本身，就像实例方法接收实例作为参数一样。要声明一个类方法，按惯例请使用以下方案：

```
class C:
    @classmethod
    def fn(cls, arg1, arg2):
        pass
```

类方法的调用可以在类上进行，也可以在实例对象上进行。通过对象调用时，**隐含的第一个参数传入的是类** 而不是对象。如果该对象包含此类以外的属性，将无法被访问。

通过派生类（或派生类的实例对象）调用基类的类方法时，**隐含的第一个参数传入的是派生类**，而不是基类，因此派生类包含的额外属性可以被访问。

注意，类方法不是其它语言中的静态方法，要创建静态方法请使用 staticmethod 装饰器。

参数

- `fn` - 要封装成类方法的函数

返回值

封装成类方法后的函数。

示例

```python
class Cat:
    name = '薛定谔'

    @classmethod
    def speak(cls):
        print(f'喵喵喵，我叫{cls.name}')

# 通过类调用
Cat.speak()

# 通过对象调用
cat = Cat()
cat.name = '狗'      # 给对象实例绑定 name 属性
cat.speak()          # 类方法访问的仍是类的 name 属性
```

推荐阅读

- Python 官方文档
- 内置函数 staticmethod
- 内置函数 property

Python 的内置函数 compile

说明：编译代码对象或 AST（抽象语法树）对象。

```
def compile(source, filename, mode, flags=0, dont_inherit=False,
optimize=-1):
    '''
    将字符串或文件编译成代码或 AST 对象

    :param source: 字符串、字节字符串，或者 AST 对象
    :param filename: 文件名或 '<string>'
    :param mode: 'exec', 'eval' 或 'single'
    :param flags: 特性
    :param dont_inherit: 是否继承
    :param optimize: 优化级别
    :return: 代码或 AST 对象
    '''
```

说明

将字符串或文件编译成代码对象或 AST（抽象语法树）对象。该对象可以被 exec 或 eval 执行。

参数

- `source` - 要编译的字符串、字节字符串，或者 AST（抽象语法树）对象
- `filename` - 文件名，如果不使用文件，而是使用 `source`，此参数通常传递 `'<string>'` 作为标识
- `mode` - `'exec'` 表示一系列语句，`'eval'` 表示单个表达式，`'single'` 表示单个交互式语句
- `flags` - 特性标志，默认为 0
- `dont_inherit` - 是否不继承，默认为 `False`
- `optimize` - 优化级别，默认为 -1，即使用解释器传入的优化选项作为优化级别

返回值

编译后的代码对象或 AST（抽象语法树）对象。

示例

```
# 编译字符串内的表达式
x = 2
obj = compile("x**10", "<string>", "eval")
print(obj)
v = eval(obj)    # 执行表达式
```

```
print(v)          # 查看结果


# 编译字符串内的代码块
code = '''
def greet(name:str):
    print('hello', name)
'''
obj = compile(code, "<string>", "exec")
exec(obj)               # 执行代码块
greet('Miss Qiu')       # 调用代码块里创建的函数


# 编译文件
obj = compile("", __file__, "exec")    # __file__ 是一个特殊的内置变量，表示当前 Python 文件的路径
print(obj)
```

推荐阅读

- Python 官方文档
- 内置函数 eval
- 内置函数 exec

Python 的内置函数 complex

说明：类型转换为 `complex`（复数）。

函数说明

```
class complex(x=0):
    '''
    类型转换为 complex

    :param x: 一个变量
    :return: 转换为 complex 后的值
    '''
```

说明

将变量转换为 `complex` 类型。

参数

- `x` - 要转换的变量

返回值

转换为 `complex` 类型后的值。

```
class complex(real=0, imag=0):
    '''
    类型转换为 complex

    :param real: 实部
    :param imag: 虚部
    :return: 转换为 complex 后的值
    '''
```

说明

通过特定的实部和虚部，创建一个类型为 `complex` 的值。也可以直接通过字面量的形式创建复数，例如 `1+2j` 其中 `j` 表示虚数单位。

参数

- `real` - 复数的实部
- `imag` - 复数的虚部

返回值

实部为 `real` 虚部为 `imag` 的复数。

示例

```
# 通过实部和虚部构造复数
x = complex(2, 3)
print(x)                # 查看复数
print(x.real)           # 查看实部
print(x.imag)           # 查看虚部
print(x.conjugate())    # 计算共轭复数
print(abs(x))           # 计算复数的模

# 通过字面量创建复数
print(2 + 3j)           # j 后缀表示虚数单位

# 其它类型转换为复数
print(complex(3.14))    # 实数转复数
print(complex('2+3j'))  # 字符串转复数，不能有空格
```

复数的属性和方法

属性/方法	说明
`real`	复数的实部
`imag`	复数的虚部
`conjugate()`	计算复数的共轭复数（实部相同，虚部为相反数）

复数的基本运算

基本运算	结果
`X.conjugate()`	`X.real - X.imag * 1j`
`abs(X)`	`(X.real**2 + X.imag**2)**0.5`
`A + B`	`(A.real + B.real) + (A.imag + B.imag) * 1j`
`A - B`	`(A.real - B.real) + (A.imag - B.imag) * 1j`

基本运算	结果
`A * B`	`(A.real * B.real - A.imag * B.imag) + (A.real * B.imag + A.imag * B.real) * 1j`
`A / B`	`((A.real * B.real + A.imag * B.imag) + (A.imag * B.real - A.real * B.imag) * 1j)/ (B.real**2 + B.imag**2)`

推荐阅读

- Python 官方文档

Python 的内置函数 delattr

说明：删除指定的属性。

函数说明

```
def delattr(obj, name:str):
    '''
    删除指定的属性

    :param obj: 一个对象
    :param name: 要删除的属性的名字
    '''
```

说明

删除对象的指定属性，等价于 `del obj.attr`。

如果属性不存在，会产生 `AttributeError` 异常。

> 请参考 hasattr、getattr 和 setattr 等相关函数。

参数

- `obj` - 要删除属性的对象
- `name` - 要删除的属性名

返回值

无

示例

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 30)
print(p.name)

# 删除name属性
delattr(p, 'name')
```

```
try:
    print(p.name)
except AttributeError as e:
    print(f"错误: {e}")  # 输出: 错误: 'Person' object has no attribute 'name'
```

推荐阅读

- Python 官方文档
- 内置函数 hasattr
- 内置函数 getattr
- 内置函数 setattr

Python 的内置函数 dict

说明：类型转换为 `dict`。

函数说明

```
class dict(**kwarg):
    '''
    类型转换为 dict

    :param kwarg: 键值对
    :return: 转换为 dict 后的值
    '''
```

说明

通过任意关键字参数创建字典，关键字参数的参数名作为键名，关键字参数的值作为值。

参数

- `kwarg` - 任意键值对参数

返回值

创建出的字典。

```
class dict(mapping, **kwarg):
    '''
    类型转换为 dict

    :param mapping: 映射对象
    :param kwarg: 键值对
    :return: 转换为 dict 后的值
    '''
```

说明

通过映射对象创建字典，可以通过任意关键字参数添加键值对。

参数

- `mapping` - 一个映射对象

- `kwarg` - 任意键值对参数

返回值

创建出的字典。

```
class dict(iterable, **kwarg):
    '''
    类型转换为 dict

    :param iterable: 包含键值对的可迭代对象
    :param kwarg: 键值对
    :return: 转换为 dict 后的值
    '''
```

说明

通过包含键值对的可迭代对象创建字典，可以通过任意关键字参数添加键值对。

参数

- `iterable` - 一个包含键值对的可迭代对象
- `kwarg` - 任意键值对参数

返回值

创建出的字典。

示例

```
# 通过键值对参数创建
print(dict(name='Alice', age=18, sex='female'))

# 通过另一个字典创建，并添加键值对
print(dict({'name':'Alice', 'age':18}, sex='female'))

# 通过可迭代对象创建（可迭代对象的元素是键值对），并添加键值对
print(dict((('name', 'Alice'), ('age', 18)), sex='female'))

# 通过 zip 组合两个列表并转换为字典
key:list[str] = ['name', 'age']
value:list = ['Alice', 18]
print(dict(zip(key, value), sex='female'))
```

注意事项

- 从 Python 3.7 开始，字典保持插入顺序；之前是无序的
- 字典键必须是不可变类型（如字符串、数字或元组）
- 当字典键是自定义对象时，需要确保正确实现 `__hash__` 和 `__eq__` 方法
- 查找、插入、删除操作的平均时间复杂度为 $O(1)$
- 字典会消耗较多内存，对于小型数据集，有时 `list` 或 `tuple` 可能更高效

推荐阅读

- Python 官方文档
- 内置函数 list
- 内置函数 tuple
- 内置函数 set
- 内置函数 frozenset

Python 的内置函数 dir

说明：返回当前本地作用域中的名称列表或对象的属性列表。

函数说明

```
def dir():
    '''
    返回当前本地作用域中的名称列表

    :return: 当前本地作用域中的名称列表
    '''
```

说明

返回当前本地作用域中的名称列表。

参数

无

返回值

当前本地作用域中的名称列表（默认按字典序排列）。

```
def dir(obj):
    '''
    返回对象的成员列表

    :param obj: 一个对象
    :return: 对象的成员列表
    '''
```

说明

通过调用对象的 `__dir__` 方法，返回对象的成员列表，包括继承的属性和方法。

参数

- `obj` - 要查看的对象

返回值

对象的成员列表（默认按字典序排列）。

示例

```
print("本地作用域中的名称列表:", dir())
print("字符串的成员列表:", dir('hello world'))
print("文件流成员列表:", dir(open(__file__)))
```

推荐阅读

- Python 官方文档
- 内置函数 vars
- 内置函数 globals
- 内置函数 locals

Python 的内置函数 divmod

说明：返回整数除法时的商和余数。

函数说明

```
def divmod(x, y):
    '''
    返回整数除法时的商和余数

    :param x: 被除数
    :param y: 除数
    :return: 商和余数的元组
    '''
```

说明

计算整数除法，并返回整除的商和余数。商向下取整，余数符号与除数相同。

除数和被除数可以不是整数：

- 如果 `x` 和 `y` 都是 `int` 类型，则商和余数均为 `int` 类型
- 覆盖 `x` 或 `y` 是 `float` 类型，则商是类型为 `float` 的整数，余数是 `float` 类型的小数

除数为 0 时会产生 `ZeroDivisionError` 异常；复数运算会产生 `TypeError` 异常。

参数

- `x` - 被除数
- `y` - 除数

返回值

商和余数组成的元组。

示例

```
print(divmod(233, 7))           # 整数计算
print(divmod(1234.5678, 7))     # 小数计算，商是整数，余数是小数

print(divmod(233, -7))          # 负数计算，商向下取整，余数符号与除数相同
print(divmod(-233, 7))          # 负数计算，商向下取整，余数符号与除数相同
```

推荐阅读

- Python 官方文档

Python 的内置函数 enumerate

说明：返回一个枚举对象。

函数说明

```
def enumerate(iterable, start=0):
    '''
    返回一个枚举对象

    :param iterable: 可迭代对象
    :param start: 起始位置
    :return: 枚举对象
    '''
```

说明

通过一个可迭代对象创建枚举对象。常用于迭代时同时获取索引（键）和值。

枚举对象是一种序列型的可迭代对象，它的元素类型是元组。 元组的第一个值是源可迭代对象元素的索引（键），第二个值是源可迭代对象元素的值。

参数

- `iterable` - 一个可迭代对象
- `start` - 起始位置，默认为 0

返回值

枚举对象。

示例

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
print(list(enumerate(seasons)))
print(list(enumerate(seasons, 2)))  # 从索引为 2 的元素开始枚举

score_list:dict[str,int] = {
    'Tom': 88,
    'Jerry': 99,
    'Spike': 66
}
```

```
# 通过枚举在迭代时同时获得键和值
for key,value in enumerate(score_list):
    print(key, value)
```

推荐阅读

- Python 官方文档
- 内置函数 dict
- 内置函数 list
- 内置函数 tuple

Python 的内置函数 eval

说明：执行表达式并返回结果。

函数说明

```
def eval(source:str|codeobject, /, globals:dict=None, locals:mapping=None):
    '''
    执行表达式并返回结果

    :param source: Python 表达式
    :param globals : 全局命名空间字典
    :param locals : 局部命名空间字典
    :return: 表达式的求值结果
    '''
```

说明

执行表达式并返回结果。

参数

- `source` - 字符串形式的 Python 表达式，或者是代码对象
- `globals` - 全局命名空间字典
  - 默认为 `None`，即使用当前环境的全局命名空间
  - 如果不包含 `__builtins__` 键，则自动插入内置模块 `builtins` 的引用作为 `__builtins__` 键的值
- `locals` - 局部命名空间字典
  - 默认为 `None`，即使用 `globals` 作为 `locals`
  - 如果 `globals` 也为 `None` 则使用当前环境的局部命名空间

返回值

表达式的执行结果。

示例

```
# 运行表达式
print(eval('1 + 1'))

# 使用当前的命名空间
x = 233
```

```
print(eval('x + 1'))

# 使用 globals 参数传入命名空间
print(eval('x + 1', globals={'x': 666}))
```

注意事项

- `eval` 可以执行任意代码，存在严重的安全风险，永远不要直接执行不可信来源的输入
- `eval` 的执行速度比直接执行代码慢，频繁地调用会显著影响性能
- 在需要限制执行环境时，应该提供 `globals` 和 `locals` 字典
- 解析数据时，更安全的替代方案包括 `ast.literal_eval()` 或 JSON

推荐阅读

- Python 官方文档
- 内置函数 exec
- 内置函数 compile

Python 的内置函数 exec

说明：执行代码。

函数说明

```
def exec(source:str|codeobject, /, globals:dict=None, locals:mapping=None, *,
closure=None):
    '''
    执行代码

    :param source: Python 代码
    :param globals : 全局命名空间
    :param locals : 局部命名空间
    :param closure : 闭包
    '''
```

说明

执行代码。

参数

- `source` - 字符串形式的 Python 代码，或者是代码对象
- `globals` - 全局命名空间字典
  - 默认为 `None`，即使用当前环境的全局命名空间
  - 如果不包含 `__builtins__` 键，则自动插入内置模块 `builtins` 的引用作为 `__builtins__` 键的值
- `locals` - 局部命名空间字典
  - 默认为 `None`，即使用 `globals` 作为 `locals`
  - 如果 `globals` 也为 `None` 则使用当前环境的局部命名空间
- `closure` - 闭包，仅在 `source` 参数是一个包含自由变量的代码对象时可用

返回值

无。

示例

```
# 执行单行代码
exec("print('hello world')")
```

```python
# 使用当前命名空间
x = 233
exec("print(x + 1)")

# 使用 globals 参数传入命名空间
exec("print(x + 1)", globals={'x': 666})

# 执行代码块
code:str = '''
def greet(name:str):
    print('hello', name)

greet(name)
'''
exec(code, globals={'name': 'Alice'})
```

注意事项

- `exec` 可以执行任意代码，存在严重的安全风险，永远不要直接执行不可信来源的输入
- `exec` 的执行速度比直接执行代码慢，频繁地调用会显著影响性能
- 在需要限制执行环境时，应该提供 `globals` 和 `locals` 字典
- 解析数据时，更安全的替代方案包括 `ast.literal_eval()` 或 JSON

推荐阅读

- Python 官方文档
- 内置函数 eval
- 内置函数 compile

Python 的内置函数 filter

说明：过滤数据。

函数说明

```
def filter(fn, iterable):
    '''
    过滤数据

    :param fn: 回调函数，返回 True 时元素保留，返回 False 时去除元素
    :param iterable: 要过滤的可迭代对象
    :return: 过滤后的可迭代对象
    '''
```

说明

过滤可迭代对象中的元素，得到新的可迭代对象。

采用惰性计算（Lazy Evaluation）的策略，也就是不会立即执行计算，而是在使用结果时才会计算相应的元素。

参数

- `fn` - 用于过滤的回调函数
  - 可迭代对象的元素会作为参数传给此函数，此函数返回 `True` 时该元素被**保留**，返回 `False` 时，该元素被**滤除**
  - 如果此参数为 `None`，则直接判断原本的数据，也就是滤除 `0`，`None`，`False` 等 假值
- `iterable` - 要过滤的可迭代对象

返回值

由过滤后保留的元素组成的可迭代对象。

示例

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(filter(lambda x : x%2 == 0, numbers))         # 元素除以 2 余数为 0 时保留，否则滤除，结果是迭代器
print(list(filter(lambda x : x%2 == 0, numbers)))   # 转换为 list
print(list(filter(None, numbers)))                  # fn 为 None，直接判断原数据
```

推荐阅读

- Python 官方文档
- 内置函数 map
- 内置函数 list
- 内置函数 tuple
- 内置函数 set
- 内置函数 frozenset
- 内置函数 dict

Python 的内置函数 float

说明：类型转换为 `float`。

函数说明

```
class float(x=0.0):
    '''
    类型转换为 float

    :param x: 一个变量
    :return: 转换为 float 后的值
    '''
```

说明

将变量转换为 `float` 类型。

参数

- `x` - 要转换的变量

返回值

转换为 `float` 后的值。

示例

```
print(float(100))               # int 转 float

print(float('100'))             # str 转 float
print(float('2e10'))            # 科学计数法
print(float('   3.14\n'))

print(float("inf"))             # 特殊值：inf（正无穷）
print(float("-inf"))            # 特殊值：-inf（负无穷）
print(float("nan"))             # 特殊值：nan（非数字）
```

推荐阅读

- Python 官方文档

Python 的内置函数 format

说明：将对象格式化为字符串。

函数说明

```
def format(obj, format_spec=''):
    '''
    类型转换为 format

    :param obj: 一个变量
    :param format_spec: 格式
    :return: 格式化后的字符串
    '''
```

说明

通过调用对象的 `__format__` 方法，将对象格式化为字符串。

参数

- `value` - 要格式化的对象
- `format_spec` - 格式规格描述，参考下方的 格式说明。

返回值

格式化后的字符串。

示例

示例：

```
# 数字格式化
print(format(1000000, ","))     # 使用逗号分隔，输出：'1,000,000'

print(format(3.14159, ".2g"))   # 保留两位有效数字，输出：'3.1'
print(format(3.14159, ".2f"))   # 保留两位小数，输出：'3.14'

print(format(5.15, ".4g"))      # 保留四位有效数字，输出：'5.15'
print(format(5.15, ".4f"))      # 保留四位小数（补0），输出：'5.1500'

# 对齐
print(format("hello", "^10"))   # 居中对齐，宽度10，输出：'  hello   '
print(format("hello", "<10"))   # 左对齐，宽度10，输出：'hello     '
```

```
print(format("hello", ">10"))   # 又对齐，宽度10，输出: '     hello'
print(format(42, "05d"))        # 用0填充，宽度5，输出: '00042'

# 填充
print(format(42, "05d"))        # 用0填充，宽度5，输出: '00042'

# 进制转换
print(format(10, "b"))          # 输出: '1010' (二进制)
print(format(255, "x"))         # 输出: 'ff' (十六进制小写)
```

格式

```
format_spec ::= [options][width][grouping]["." precision][type]
options     ::= [[fill]align][sign]["z"]["#"]["0"]
fill        ::= <any character>
align       ::= "<" | ">" | "=" | "^"
sign        ::= "+" | "-" | " "
width       ::= digit+
grouping    ::= "," | "_"
precision   ::= digit+
type        ::= "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g"
              | "G" | "n" | "o" | "s" | "x" | "X" | "%"
```

对齐选项

选项	说明
`<`	左对齐（默认）
`>`	右对齐（数值类型默认）
`=`	填充对齐
`^`	居中对齐

符号选项

选项	说明
`+`	正负数均显示符号
`-`	仅负数显示负号（默认）
`(空格)`	正数添加前导空格，负数显示负号

分隔选项

选项	说明
`,`	使用逗号分隔，默认每 3 位数插入一个分隔符
`_`	使用下划线分隔，默认每 3 位数插入一个分隔符

整数选项

选项	说明
`b`	二进制
`d`	十进制
`o`	八进制
`x`	十六进制，字母小写
`X`	十六进制，字母大写
`n`	同 `d`，但使用分隔符时按照当前语言区域插入，例如中文每 4 位数插入一个分隔符
`c`	在打印之前将整数转换为相应的unicode字符

浮点数选项

选项	说明
`e`	科学计数法，字母小写
`E`	科学计数法，字母大写
`f`	定点表示法，默认 6 位小数，字母小写
`F`	定点表示法，默认 6 位小数，字母大写
`g`	常规格式，字母小写
`G`	常规格式，字母大写
`n`	同 `g`，但使用分隔符时按照当前语言区域插入，例如中文每 4 位数插入一个分隔符
`%`	百分比，数字将乘以 100 并显示为 `f` 格式，后面添加百分号

推荐阅读

- Python 官方文档
- 字符串格式化

Python 的内置函数 frozenset

说明：类型转换为 `frozenset`。

函数说明

```
def frozenset(x):
    '''
    类型转换为 frozenset

    :param x: 一个变量
    :return: 转换为 frozenset 后的值
    '''
```

说明

将变量类型转换为不可变集合 `frozenset`。

参数

- `x` - 要转换的值

返回值

转换为 `frozenset` 后的值

示例

```
print(frozenset({1, 2, 3}))
```

推荐阅读

- Python 官方文档
- Python 的容器类型-集合
- 内置函数 set
- 内置函数 dict
- 内置函数 list
- 内置函数 tuple

Python 的内置函数 getattr

说明：获取属性的值。

函数说明

```
def getattr(obj, name:str):
    '''
    获取属性的值

    :param obj: 一个对象
    :param name: 属性的名字
    '''
```

说明

获取对象属性的值。

1. 首先调用 `__getattribute__` 方法，尝试获取对象的 `__dict__` 属性字典
2. 未找到该属性时，如果对象实现了 `__getattr__` 方法，则会调用该方法

属性不存在时，产生 `AttributeError` 异常。

> `__getattribute__` 是 `object` 中定义的方法，默认读取 `__dict__` 属性字典。 如果 `__dict__` 属性字典中没有指定的属性，则尝试调用 `__getattr__` 方法。
>
> - 重载 `__getattr__` 方法可以自定义查找 `__dict__` 属性字典中没有的属性的方式。
> - 重载 `__getattribute__` 方法可以自定义所有属性的查找方式。

> 请参考 hasattr、delattr 和 setattr 等相关函数。

参数

- `obj` - 要获取属性的对象
- `name` - 要获取的属性名

返回值

属性的值；如果属性不存在，则产生 `AttributeError` 异常。

示例

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 30)
print(p.name)

print(getattr(p, 'name'))
```

推荐阅读

- Python 官方文档
- 内置函数 setattr
- 内置函数 hasattr
- 内置函数 delattr

Python 的内置函数 globals

说明：返回实现当前模块命名空间的字典。

函数说明

```
def globals():
    '''
    返回实现当前模块命名空间的字典

    :return: 当前模块命名空间的字典
    '''
```

说明

返回实现当前模块全局命名空间的字典。

可以编辑返回的字典实现对全局变量的修改。

参数

无

返回值

当前模块全局命名空间的字典。

示例

```
# 查看全局变量
print(globals())

# 创建全局变量
globals()['name'] = 'Alice'
print(name)

# 修改全局变量
globals()['name'] = 'Dorothy'
print(name)
```

推荐阅读

- Python 官方文档
- 内置函数 locals
- 内置函数 dir
- 内置函数 vars

Python 的内置函数 hasattr

说明：判断属性是否存在。

函数说明

```
def hasattr(obj, name:str):
    '''
    判断对象的属性是否存在

    :param obj: 一个对象
    :param name: 属性的名字
    :return: 属性存在则返回 True，否则返回 False
    '''
```

说明

判断对象的属性是否存在。

1. 首先调用 `__getattribute__` 方法，尝试获取对象的 `__dict__` 属性字典
2. 未找到该属性时，如果对象实现了 `__getattr__` 方法，则会调用该方法

> `__getattribute__` 是 `object` 中定义的方法，默认读取 `__dict__` 属性字典。 如果 `__dict__` 属性字典中没有指定的属性，则尝试调用 `__getattr__` 方法。
>
> - 重载 `__getattr__` 方法可以自定义 `__dict__` 属性字典中没有的属性的查找方式。
> - 重载 `__getattribute__` 方法可以自定义所有属性的查找方式。

> 请参考 delattr、getattr 和 setattr 等相关函数。

参数

- `obj` - 要判断的对象
- `name` - 要判断的属性名

返回值

- 如果属性存在，则返回 `True`
- 如果属性不存在，则返回 `False`

示例

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 30)
print(hasattr(p, 'name'))
print(hasattr(p, 'age'))
print(hasattr(p, 'sex'))
```

推荐阅读

- Python 官方文档
- 内置函数 delattr
- 内置函数 getattr
- 内置函数 setattr

Python 的内置函数 hash

说明：获取哈希值。

函数说明

```
def hash(obj):
    '''
    获取哈希值

    :param obj: 一个对象
    :return: 对象的哈希值
    '''
```

说明

通过调用对象的 `__hash__` 方法，获取对象的哈希值。这个值被用作 `dict`，`set` 和 `frozenset` 的唯一键。

参数

- `obj` - 要获取哈希值的对象

返回值

对象的哈希值。

示例

```
print(hash(0))
print(hash(233))
print(hash(3.1415926))
print(hash(''))
print(hash('hello world'))
```

推荐阅读

- Python 官方文档
- 内置函数 dict
- 内置函数 set

- 内置函数 frozenset

Python 的内置函数 help

说明：启动内置的帮助系统。

函数说明

```
def help():
    '''
    启动内置的帮助系统
    '''
```

说明

启动内置的交互式帮助系统。

参数

无

返回值

无

```
def help(text:str):
    '''
    在帮助系统中搜索文本

    :param text: 要搜索的字符串
    '''
```

说明

在帮助系统中搜索文本。

参数

- `text` - 要搜索的字符串

返回值

无

```
def help(obj):
    '''
    查看对象的帮助信息

    :param obj: 要查看帮助的信息对象
    '''
```

说明

查看对象的帮助信息。

参数

- `obj` - 要查看帮助的信息对象

返回值

无

示例

```
help(print)
```

推荐阅读

- Python 官方文档

Python 的内置函数 hex

说明：将一个整数转换为带前缀 `0x` 的十六进制字符串。

函数说明

```
def hex(x:int):
    '''
    将一个整数转换为带前缀 `0x` 的十六进制字符串

    :param x: 一个整数
    :return: x 的十六进制字符串形式
    '''
```

说明

将一个整数转换为带 `0x` 前缀的十六进制字符串。

如果参数 `x` 不是整数，则它必须定义 `__index__` 方法，该返回一个整数， `hex` 会将其返回的整数转换为带 `0x` 前缀的十六进制字符串。

参数

- `x` - 要转换的整数

返回值

以 `0x` 开头的十六进制字符串。

示例

```
print('0 =>', hex(0))
print('10 =>', hex(10))
print('1024 =>', hex(1024))

class Custom:
    def __index__(self) -> int:
        return 1024

obj = Custom()
print('obj =>', hex(obj))
```

推荐阅读

- Python 官方文档
- 内置函数 bin
- 内置函数 oct

Python 的内置函数 id

说明：返回对象的 ID。

函数说明

```
def id(obj):
    '''
    返回对象的 ID

    :param obj: 一个对象
    :return: 对象的 ID
    '''
```

说明

返回对象的 ID，通常为该对象在内存中的地址。

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。

参数

- `x` - 要获取 ID 的对象

返回值

对象的 ID。

示例

```
print(id(65535))
print(id('hello world'))
print(id([1, 2, 3, 4, 5]))
print(id(object()))
```

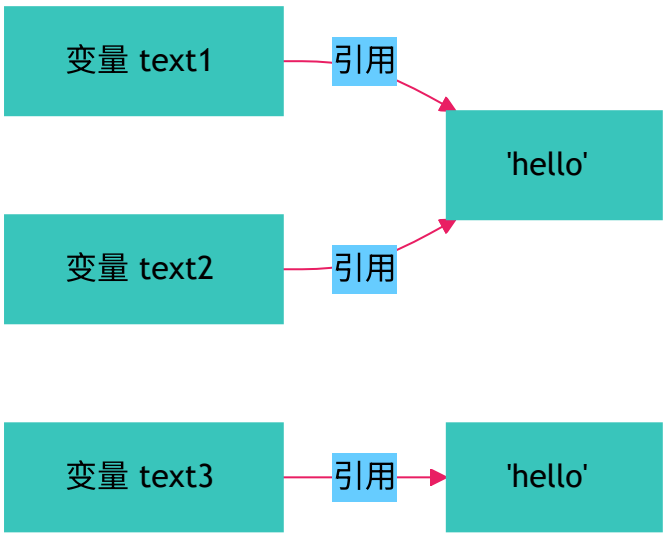
注意事项

两个值相同的变量 **可能** 具有相同的 ID，因为它们可能引用同一个内存地址；但是在开发中不应依赖此特性。

例如：

```python
# 值相同且 ID 相同
text1:str = 'hello'
text2:str = 'hello'
print(f"{text1}({id(text1)})", f"{text2}({id(text2)})", id(text1) == id(text2))

# 值相同但 ID 不同
text3:str = 'hello world'[:5]
print(f"{text1}({id(text1)})", f"{text3}({id(text3)})", id(text1) == id(text3))
```



推荐阅读

- Python 官方文档

Python 的内置函数 input

说明：获取输入。

函数说明

```
def input(prompt='') -> str:
    '''
    获取输入

    :param prompt: 提示文本
    :return: 输入的字符串
    '''
```

说明

打印提示信息并获取输入。

如果读取到 EOF，则产生 `EOFError` 异常。

如果加载了 `readline` 模块，`input` 将使用它来提供复杂的行编辑和历史记录功能。

参数

- `prompt` - 要打印的提示信息，默认为空

返回值

输入的字符串。

示例

```
text = input('请输入：')
print('输入的是：', text)
```

推荐阅读

- Python 官方文档
- 内置函数 print

Python 的内置函数 int

说明：类型转换为 `int`。

函数说明

```
class int(x=0):
    '''
    类型转换为 int

    :param x: 一个变量
    :return: 转换为 int 后的值
    '''
```

说明

将变量转换为 `int` 类型。

特别地：

1. 如果参数定义了 `__int__` 方法，将以该方法的返回值作为转换结果。
2. 如果参数定义了 `__index__` 方法，将以该方法的返回值作为转换结果。
3. 如果参数定义了 `__trunc__` 方法，将以该方法的返回值作为转换结果。

参数

- `x` - 要转换的变量，默认为 0

返回值

转换为 `int` 后的值。

```
class int(text:str|bytes|bytearray, /, base=10):
    '''
    将字符串转换为整数

    :param text: 要转换的字符串
    :param base: 字符串的进制
    :return: 转换为 int 后的值
    '''
```

说明

将字符串转换为整数。

特别地：

1. 如果参数定义了 `__int__` 方法，将以该方法的返回值作为转换结果。
2. 如果参数定义了 `__index__` 方法，将以该方法的返回值作为转换结果。
3. 如果参数定义了 `__trunc__` 方法，将以该方法的返回值作为转换结果。

参数

- `text` - 转换的字符串
- `base` - 字符串中数值的进制

返回值

转换为 `int` 后的值。

示例

```
print(int(3.1415926))
print(int('100'))           # 默认采用十进制
print(int('100', base=2))   # 采用二进制
```

推荐阅读

- Python 官方文档

Python 的内置函数 isinstance

说明：判断是否是某个类型的实例。

函数说明

```
def isinstance(obj, classinfo) -> bool:
    '''
    获取输入

    :param obj: 一个对象
    :param classinfo: 一个类型
    :return: 如果 obj 是 classinfo 类型则返回 True，否则返回 False
    '''
```

说明

判断是否是某个类型的实例。

`classinfo` 可以是多个类构成的元组或 `union` 类型， `obj` 只要是其中一个类的实例即返回 `True` 。

参数

- `obj` - 要判断的对象
- `classinfo` - 要判断的类

返回值

- 如果 `obj` 是 `classinfo` 或其子类的实例，则返回 `True`
- 如果 `obj` 不是 `classinfo` 或其子类类的实例，则返回 `True`

示例

```
print(isinstance(233, int))             # True
print(isinstance(233, float))           # False
print(isinstance('233', int))           # False
print(isinstance('233', int|str))       # True，union 类型满足其一
print(isinstance('233', (int, str)))    # True，元组满足其一

class Animal:           # 动物
    pass

class Mammal(Animal):   # 哺乳动物
```

```
    pass

class Cat(Mammal):      # 猫
    pass

print(isinstance(Cat(), Cat))           # True，Cat() 是 Cat 的实例
print(isinstance(Cat(), Mammal))        # True，Cat() 是 Mammal 的实例
print(isinstance(Cat(), Animal))        # True，Cat() 是 Animal 的实例
print(isinstance(Mammal(), Cat))        # False，Mammal() 不是 Cat 的实例
print(isinstance(Mammal(), Animal))     # True，Mammal() 是 Animal 的实例
```

推荐阅读

- Python 官方文档
- 内置函数 issubclass
- 内置函数 type
- 内置函数 super

Python 的内置函数 issubclass

说明：判断是否是某个类的子类。

函数说明

```
def issubclass(cls, classinfo) -> bool:
    '''
    获取输入

    :param cls: 一个类
    :param classinfo: 一个类型
    :return: 如果 cls 是 classinfo 的子类则返回 True，否则返回 False
    '''
```

说明

判断一个类是否是另一个类的子类（直接或间接继承）。

`classinfo` 可以是多个类构成的元组或 `union` 类型，`cls` 只要是其中一个类的子类即返回 `True`。

`cls` 和 `classinfo` 是相同的类时，也返回 `True`。

参数

- `cls` - 要判断的子类
- `classinfo` - 要判断的父类

返回值

- 如果 `cls` 是 `classinfo` 的子类，则返回 `True`
- 如果 `cls` 不是 `classinfo` 的子类，则返回 `True`

示例

```
print(issubclass(bytes, str))           # False
print(issubclass(int, float))           # False
print(issubclass(int, object))          # True，object 是所有类型的终极父类

class Animal:                           # 动物
    pass

class Mammal(Animal):                   # 哺乳动物
```

```
    pass

class Dog(Mammal):                      # 狗
    pass

# 简单继承检查
print(issubclass(Dog, Mammal))          # True，直接继承
print(issubclass(Dog, Animal))          # True，间接继承
print(issubclass(Dog, Dog))             # True，相同类

# 多个类
print(issubclass(Dog, str|Mammal))      # True，union 类型满足其一
print(issubclass(Dog, (str, Mammal)))   # True，元组满足其一

# 非继承关系
print(issubclass(Dog, str))             # False
```

推荐阅读

- Python 官方文档
- 内置函数 isinstance
- 内置函数 type
- 内置函数 super

Python 的内置函数 iter

说明：获取可迭代对象的迭代器。

函数说明

```
def iter(iterable):
    '''
    获取可迭代对象的迭代器

    :param iterable: 一个可迭代对象
    :return: 参数的迭代器
    '''
```

说明

通过调用该对象的 `__iter__` 方法，获取可迭代对象的迭代器。

参数

- `iterable` - 要获取迭代器的可迭代对象

返回值

可迭代对象的迭代器

```
def iter(iterable, sentinel):
    '''
    获取可迭代对象的迭代器

    :param iterable: 一个可迭代对象
    :param sentinel: 哨兵值
    :return: 参数的迭代器
    '''
```

说明

创建一个特殊迭代器，不会调用可迭代对象的 `__iter__` 方法。

`iterable` 参数必须是一个可调用对象，每次调用迭代器的 `__next__` 方法时，都会不带参数地调用 `iterable`，并使用该返回值作为 `__next__` 的返回值。

如果返回值等于 `sentinel`，则会产生 `StopIteration` 异常，用于通知迭代结束。

参数

- `iterable` - 要获取迭代器的可迭代对象
- `sentinel` - 哨兵值

返回值

可迭代对象的迭代器

示例

常规的迭代器与可迭代对象：

```
# 迭代器
class Iterator:
    def __init__(self, stop):
        self.__stop = stop
        self.__current = 0

    def __next__(self):
        if self.__current < self.__stop:
            self.__current += 1
            return self.__current - 1
        else:
            raise StopIteration # 产生 StopIteration 异常通知迭代结束

# 可迭代对象
class Iterable:
    def __init__(self, stop):
        self.__iterator = Iterator(stop)

    def __iter__(self):
        return self.__iterator

# 创建可迭代对象
iterable = Iterable(10)

# 获取迭代器，没有哨兵值
iterator = iter(iterable)
print(iterator)

# 迭代
while (n := next(iterator, None)) != None:
    print(n)
```

可调用的可迭代对象与哨兵值：

```python
# 可迭代对象
class Iterable:
    def __init__(self):
        self.__current = 0

    # 使用 __call__ 代替迭代器
    def __call__(self):
        self.__current += 1
        return self.__current - 1

# 创建可迭代对象
iterable = Iterable()

# 获取迭代器，哨兵值为 5
iterator = iter(iterable, 5)
print(iterator)

# 迭代
while (n := next(iterator, None)) != None:
    print(n)
```

推荐阅读

- Python 官方文档
- 迭代器和可迭代对象
- 内置函数 next
- 内置函数 aiter
- 内置函数 anext

Python 的内置函数 len

说明：获取长度。

函数说明

```
def len(obj):
    '''
    获取长度

    :param obj: 一个对象
    :return: 对象的长度
    '''
```

说明

通过调用对象的 `__len__` 方法获取长度。

如果获取的长度大于 `sys.maxsize`，会产生 `OverflowError` 异常。

参数

- `obj` - 要获取长度的对象

返回值

对象的长度

示例

```
print(len('hello world'))                   # 获取字符串的长度
print(len((1, 2, 3, 4, 5)))                 # 获取元组的长度
print(len([1, 2, 3, 4, 5]))                 # 获取列表的长度
print(len({'Cat', 'Dog'}))                  # 获取集合的长度
print(len({'name': 'Alice', 'age': 17}))    # 获取字典的长度
print(len(range(5)))                        # 获取可迭代对象的长度
```

推荐阅读

- Python 官方文档

Python 的内置函数 list

说明：类型转换为 `list`。

函数说明

```
class list():
    '''
    创建一个空的列表

    :return: 空列表
    '''
```

说明

创建一个空的列表，等同于 `[]`。

参数

无

返回值

空的列表

```
class list(iterable):
    '''
    类型转换为 list

    :param iterable: 一个可迭代对象
    :return: 转换为 list 后的值
    '''
```

说明

将变量转换为 `list` 类型

参数

- `iterable` - 要转换的可迭代对象

返回值

转换为 `list` 后的列表

示例

```
print(list((1, 2, 3, 4, 5)))                        # 元组转列表
print(list({'Tom', 'Jerry', 'Spike', 'Tuffy'}))     # 集合转列表
print(list({'name':'Alice', 'age':18}))             # 字典转列表，只保留键
print(list(range(5)))                               # 可迭代对象转列表
```

推荐阅读

- Python 官方文档
- 内置函数 tuple
- 内置函数 dict
- 内置函数 set
- 内置函数 frozenset
- 内置函数 slice

Python 的内置函数 locals

说明：返回一个代表当前局部符号表的映射对象。

函数说明

```
def locals():
    '''
    返回一个代表当前局部符号表的映射对象

    :return: 当前局部符号表的映射对象
    '''
```

说明

返回实现当前局部命名空间的字典副本。

对该副本的修改不会实际影响到局部命名空间。

在全局作用域中调用 `locals` 会返回 `globals` 返回的字典。

参数

无

返回值

当前局部符号表

示例

```
def calculation(x, y):
    temp = x * y
    print(locals())     # 查看当前所有局部变量
    return temp + 100

calculation(1, 2)
```

推荐阅读

- Python 官方文档

- 内置函数 globals
- 内置函数 dir
- 内置函数 vars

Python 的内置函数 map

说明：将可迭代对象进行映射。

函数说明

```
def map(fn, iterable, *iterables):
    '''
    将可迭代对象进行映射

    :param fn: 回调函数，返回元素映射后的值
    :param iterable: 要映射的可迭代对象
    :param iterables: 可变长度的可迭代对象
    :return: 映射后的可迭代对象
    '''
```

说明

将可迭代对象进行映射。

回调函数的参数与 `map` 的可迭代对象参数数量及顺序保持一致。如果多个可迭代对象参数的长度不同，按最短的长度进行映射。

采用惰性计算（Lazy Evaluation）的策略，也就是不会立即执行计算，而是在使用结果时才会计算相应的元素。

参数

- `fn` - 映射的回调函数，可迭代对象的元素会作为参数传给此函数，此函数的返回值作为映射后的值
- `iterable` - 要映射的可迭代对象
- `iterables` - 任意数量的可迭代对象

返回值

映射后的 `map` 对象。

示例

```
# 单个可迭代对象
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(map(lambda x : x**2, numbers))
print(list(map(lambda x : x**2, numbers)))
```

```
# 多可迭代对象
numbers1 = [1, 2, 3, 4, 5]
numbers2 = [2, 4, 6, 8]          # 长度不同，按最短的处理
print(list(map(lambda x,y : x*y, numbers1, numbers2)))
```

推荐阅读

- Python 官方文档
- 内置函数 filter
- 内置函数 list
- 内置函数 tuple
- 内置函数 set
- 内置函数 frozenset
- 内置函数 dict

Python 的内置函数 max

说明：获取最大值。参考 min 函数。

函数说明

```
def max(iterable, *, key=None):
    '''
    获取最大值

    :param iterable: 一个可迭代对象
    :param key: 回调函数，返回用于比较的字段
    :return: 可迭代对象的最大值
    '''
```

说明

获取最大值。

参数

- `iterable` - 一个可迭代对象
- `key` - 回调函数，可迭代对象的元素都会传递给此函数，通过此函数的返回值进行比较，默认为 `None`，即直接比较原本的值

返回值

最大的一个值；可迭代对象为空时产生 `ValueError` 异常。

```
def max(iterable, *, default, key=None):
    '''
    获取最大值

    :param iterable: 一个可迭代对象
    :param default: 默认值，可迭代对象为空时返回这个值
    :param key: 回调函数，返回用于比较的字段
    :return: 可迭代对象的最大值
    '''
```

说明

获取最大值。

参数

- `iterable` - 一个可迭代对象
- `default` - 默认值，可迭代对象为空时返回这个值
- `key` - 回调函数，可迭代对象的元素都会传递给此函数，通过此函数的返回值进行比较，默认为 `None`，即直接比较原本的值

返回值

最大的一个值

```
def max(arg1, arg2, *args, key=None):
    '''
    获取最大值

    :param arg1: 一个值
    :param arg2: 一个值
    :param args: 任意个值
    :param key: 回调函数，返回用于比较的字段
    :return: 最大值
    '''
```

说明

获取最大值。

参数

- `arg1` - 一个值
- `arg2` - 一个值
- `args` - 任意个值
- `key` - 回调函数，参与比较的值都会传递给此函数，通过此函数的返回值进行比较，默认为 `None`，即直接比较原本的值

返回值

最大的一个值

示例

```
print(max(2, 8, 4, 3, 9, 1, 6, 5, 7))       # 多个参数中取最大值
print(max([2, 8, 4, 3, 9, 1, 6, 5, 7]))     # 可迭代对象中取最大值
print(max([], default=-1))                  # 设置默认值为 -1
```

```python
# 设置回调函数，指定比较 x[1] 的大小
print(max([('Tom', 8), ('Jerry', 9), ('Spike', 5), ('Tuffy', 3)], key=lambda x:x[1]))
```

推荐阅读

- Python 官方文档
- 内置函数 min

Python 的内置函数 memoryview

说明：返回由给定实参创建的“内存视图”对象。

函数说明

```
def memoryview(obj):
    '''
    返回由给定实参创建的“内存视图”对象

    :param obj: 一个对象
    :return: 由给定实参创建的“内存视图”对象
    '''
```

说明

由给定实参创建的“内存视图”对象，它在不复制底层数据的情况下直接操作原始数据。这在处理大型二进制数据（如音频、视频或图像文件）时特别有用，可以显著提升性能并减少内存消耗。

参数

- `obj` - 用于创建内存视图的对象，该对象需要支持缓冲区协议

返回值

对象的内存视图。

示例

```
view = memoryview('甲乙丙丁戊己庚辛壬癸'.encode())
print(view)
print(view[2])
```

推荐阅读

- Python 官方文档
- 内存视图
- 缓冲区协议
- 内置函数 bytearray
- 内置函数 bytes

Python 的内置函数 min

说明：获取最小值。参考 max 函数。

函数说明

```
def min(iterable, *, key=None):
    '''
    获取最小值

    :param iterable: 一个可迭代对象
    :param key: 回调函数，返回用于比较的字段
    :return: 可迭代对象的最小值
    '''
```

说明

获取最小值。

参数

- `iterable` - 一个可迭代对象
- `key` - 回调函数，可迭代对象的元素都会传递给此函数，通过此函数的返回值进行比较，默认为 `None`，即直接比较原本的值

返回值

最小的一个值；可迭代对象为空时产生 `ValueError` 异常。

```
def min(iterable, *, default, key=None):
    '''
    获取最小值

    :param iterable: 一个可迭代对象
    :param default: 默认值，可迭代对象为空时返回这个值
    :param key: 回调函数，返回用于比较的字段
    :return: 可迭代对象的最小值
    '''
```

说明

获取最小值。

参数

- `iterable` - 一个可迭代对象
- `default` - 默认值，可迭代对象为空时返回这个值
- `key` - 回调函数，可迭代对象的元素都会传递给此函数，通过此函数的返回值进行比较，默认为 `None`，即直接比较原本的值

返回值

最小的一个值

```
def min(arg1, arg2, *args, key=None):
    '''
    获取最小值

    :param arg1: 一个值
    :param arg2: 一个值
    :param args: 任意个值
    :param key: 回调函数，返回用于比较的字段
    :return: 最小值
    '''
```

说明

获取最小值。

参数

- `arg1` - 一个值
- `arg2` - 一个值
- `args` - 任意个值
- `key` - 回调函数，参与比较的值都会传递给此函数，通过此函数的返回值进行比较，默认为 `None`，即直接比较原本的值

返回值

最小的一个值

示例

```
print(min(2, 8, 4, 3, 9, 1, 6, 5, 7))       # 多个参数中取最小值
print(min([2, 8, 4, 3, 9, 1, 6, 5, 7]))     # 可迭代对象中取最小值
print(min([], default=-1))                  # 设置默认值为 -1
```

```python
# 设置回调函数，指定比较 x[1] 的大小
print(min([('Tom', 8), ('Jerry', 9), ('Spike', 5), ('Tuffy', 3)], key=lambda x:x[1]))
```

推荐阅读

- Python 官方文档
- 内置函数 max

Python 的内置函数 next

说明：获取迭代器的下一数据项（调用对象的 `__next__` 方法）。

函数说明

```
def next(iterator):
    '''
    获取迭代器的下一数据项，
    没有下一项时产生 StopIteration 异常

    :param iterator: 一个迭代器
    :return: 迭代器的下一项
    '''
```

说明

通过调用对象的 `__next__` 方法，获取异步迭代器的下一数据项。

异步迭代器没有下一项时，应当从 `__next__` 方法中通过抛出 `StopIteration` 异常进行通知， `next` 函数会将该异常继续抛出。

参数

- `iterator` - 一个可迭代对象

返回值

迭代器的下一项数据；如果迭代器没有下一项，则抛出 `StopIteration` 异常。

```
def next(iterator, default):
    '''
    获取迭代器的下一数据项，没有下一项时返回 default

    :param iterator: 一个迭代器
    :param default: 默认值
    :return: 迭代器的下一项
    '''
```

说明

通过调用对象的 `__next__` 方法，获取异步迭代器的下一数据项。

异步迭代器没有下一项时，应当从 `__next__` 方法中通过抛出 `StopIteration` 异常进行通知，`next` 函数会返回 `default` 参数传入的默认值。

参数

- `iterator` - 一个可迭代对象
- `default` - 迭代器没有下一项时返回的默认值

返回值

迭代器的下一项数据；如果迭代器没有下一项，则返回 `default` 参数传入的默认值。

示例

```
# 迭代器类
class Iterator:
    def __init__(self, stop):
        self.__stop = stop
        self.__current = 0

    def __next__(self):
        if self.__current < self.__stop:
            self.__current += 1
            return self.__current - 1
        else:
            raise StopIteration

# 迭代器对象
iterator = Iterator(10)

# 迭代
while (value := next(iterator, None)) is not None:
    print(value)
```

推荐阅读

- Python 官方文档
- 迭代器和可迭代对象
- 内置函数 iter
- 内置函数 aiter
- 内置函数 anext

Python 的内置函数 object

说明：所有类的终极基类，调用时构建一个基本对象。

函数说明

```
class object():
    '''
    所有类的终极基类，调用时构建一个基本对象

    :return: 一个基本对象
    '''
```

说明

所有类的终极基类，调用时构建一个基本对象。

直接通过 `object` 构建的基本对象不包含任何属性，也无法通过 setattr 添加属性。

参数

无

返回值

一个基本对象。

```
print(object())
```

推荐阅读

- Python 官方文档
- 面向对象

Python 的内置函数 oct

说明：将一个整数转换为带前缀 `0o` 的八进制字符串。

函数说明

```
def oct(x:int):
    '''
    将一个整数转换为带前缀 `0o` 的八进制字符串

    :param x: 一个整数
    :return: x 的八进制字符串形式
    '''
```

说明

将一个整数转换为带 `0o` 前缀的八进制字符串。

如果参数 `x` 不是整数，则它必须定义 `__index__` 方法，该返回一个整数， `oct` 会将其返回的整数转换为带 `0o` 前缀的八进制字符串。

参数

- `x` - 要转换的整数

返回值

以 `0o` 开头的八进制字符串。

示例

```
print('0 =>', oct(0))
print('10 =>', oct(10))
print('1024 =>', oct(1024))

class Custom:
    def __index__(self) -> int:
        return 1024

obj = Custom()
print('obj =>', oct(obj))
```

推荐阅读

- Python 官方文档
- 内置函数 bin
- 内置函数 hex

Python 的内置函数 open

说明：打开文件。

函数说明

```
def open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None,
closefd=True, opener=None):
    '''
    打开文件

    :param file: 文件路径或描述符
    :param mode: 模式
    :param buffering: 缓冲策略
    :param encoding: 编码格式，默认使用本地编码
    :param errors: 指定如何处理编解码错误
    :param newline: 如何解析换行符
    :param closefd: 如果 file 时文件描述符，此参数决定关闭文件流时是否自动关闭描述符
    :param opener: 自定义开启器
    :return: 文件对象
    '''
```

说明

打开文件。

如果该文件不能被打开，则产生 `OSError` 异常。

参数

- `file` - 文件路径或描述符
- `mode` - 打开的模式
- `buffering` - 缓冲策略
- `encoding` - 文本模式下指定编码，默认为 `None`，使用操作系统的编码
- `errors` - 指定如何处理编解码错误，默认为 `None`
- `newline` - 指定如何处理换行符，默认为 `None`
- `closefd` - 如果 file 时文件描述符，此参数决定关闭文件流时是否自动关闭描述符，默认为 `True`
- `opener` - 自定义开启器，通过此函数打开文件，默认为 `None`

返回值

用于读写文件的文件流。

文件的打开模式

- `r` 表示读，如果文件不存在则会报错
- `w` 表示写，如果文件已经存在则会清空原文件
- `a` 表示追加，打开文件时偏移量在文件末尾
- `+` 表示读写
- `b` 表示二进制模式，常用于非文本文件，例如图片、可执行文件等

模式	说明	文件存在	文件不存在	初始偏移	返回值类型
`r`	只读	打开文件	报错 `FileNotFoundError`	文件开头	`str`
`w`	只写	清空文件	新建文件	文件开头	`str`
`a`	追加写	打开文件	新建文件	文件末尾	`str`
`r+`	读写	打开文件	新建文件	文件开头	`str`
`w+`	读写	清空文件	新建文件	文件开头	`str`
`a+`	追加读写	打开文件	新建文件	文件末尾	`str`
`rb`	只读	打开文件	报错 `FileNotFoundError`	文件开头	`byets`
`wb`	只写	清空文件	新建文件	文件开头	`byets`
`ab`	追加写	打开文件	新建文件	文件末尾	`byets`
`rb+`	读写	打开文件	报错 `FileNotFoundError`	文件开头	`byets`
`wb+`	读写	清空文件	新建文件	文件开头	`byets`
`ab+`	追加读写	打开文件	新建文件	文件末尾	`byets`

缓冲策略

- 二进制模式下，可以传入 0 来关闭缓冲区
- 二进制模式下，可以传入一个大于 1 的整数表示缓冲区的字节大小
- 文本模式下，可以传入 1，将缓冲策略设为 **行缓冲**

编码错误的处理方式

- `None` 或 `'strict'` - 编码错误时产生 `ValueError` 异常
- `'ignore'` - 忽略错误，通常会丢失数据
- `'replace'` - 使用 `'?'` 替换有错误的位置
- `'surrogateescape'` - 使用 `U+DC08` 至 `U+DCFF` 之间的值表示不正确的字节

- `'xmlcharrefreplace'` - 编码格式不支持的字符将被替换为相应的 XML 字符引用，仅在写入时可用
- `'backslashreplace'` - 使用反斜杠转义表示错误的数据
- `'namereplace'` - 使用 `\N{...}` 转义表示错误的数据

换行符的处理

- `None` - 采取通用换行模式，`'\n'`，`'\r'`，`'\r\n'` 都会被替换为 `'\n'` 返回
- `''` - 采取通用换行模式，换行符不会被替换，而是直接返回
- `'\n'` - 仅在 `'\n'` 换行
- `'\r'` - 仅在 `'\r'` 换行
- `'\r\n'` - 仅在 `'\r\n'` 换行

示例

```
# __file__ 是一个特殊的内置变量，表示当前 Python 文件的路径
with open(__file__, encoding='utf-8') as fp:
    print(fp)               # 打印文件对象
    print(fp.read())        # 读取文件内容并打印
```

推荐阅读

- Python 官方文档

Python 的内置函数 ord

说明：单个 Unicode 字符的字符串转整数编码。

函数说明

```
def ord(x:str):
    '''
    单个 Unicode 字符的字符串转整数编码

    :param x: 单个 Unicode 字符的的字符串
    :return: 单个 Unicode 字符的编码
    '''
```

说明

将单个 Unicode 字符字符串转换为整数编码。

如果参数不是单个字符，将产生 `TypeError` 异常

反向的转换请使用 chr 函数。

参数

- `x` - 单个 Unicode 字符的的字符串

返回值

`x` 的 Unicode 编码；如果参数不是单个字符，将产生 `TypeError` 异常

示例

```
# ASCII 兼容字符
print('A 的编码是', ord('A'))
print('B 的编码是', ord('B'))

# 汉字
print('甲 的编码是', ord('甲'))
print('我 的编码是', ord('我'))

# emoji
print('😀 的编码是', ord('😀'))
print('🤪 的编码是', ord('🤪'))
```

推荐阅读

- Python 官方文档
- 内置函数 chr

Python 的内置函数 pow

说明：计算乘方。

函数说明

```
def pow(x, y, mod=None):
    '''
    获取长度

    :param x: 底数
    :param y: 指数
    :param mod: 取余
    :return: x 的 y 次方
    '''
```

说明

计算 `x` 的 `y` 次方，相当于 `x**y`。

如果 `mod` 不为 `None`，则将结果除以 `mod` 取余数。

有以下公式：

倒数次幂：$x^{\frac{1}{n}} = \sqrt[n]{x}$

负数次幂：$x^{-n} = \frac{1}{x^n}$

参数

- `x` - 底数
- `y` - 指数
- `mod` - 默认为 `None`，如果不为 `None`，则将结果除以 `mod` 取余数

返回值

`x` 的 `y` 次方

示例

```
print(pow(2, 3))        # 2 的 3 次方是 8；相当于 2**3
print(pow(2, 3, 5))     # 2 的 3 次方是 8，8 除以 5 得 1 余 3；相当于 2**3 % 5
```

```
print(pow(8, 1/3))      # 8 的 1/3 次方，也就是 8 开 3 次方，结果是是 2；相当于 8**(1/2)
```

推荐阅读

- Python 官方文档

Python 的内置函数 print

说明：打印。

函数说明

```
def print(*objs, sep=' ', end='\n', file=None, flush=False):
    '''
    获取长度

    :param objs: 任意数量要打印的对象
    :param sep: 间隔字符，默认空格
    :param end: 末尾字符，默认换行
    :param file: 打印的目标（一个具有 write(string) 方法的对象），默认为标准输出
    :param flush: 是否强制刷新
    '''
```

说明

打印文本。

参数

- `objs` - 任意数量要打印的对象，会通过 `__str__` 方法转换为字符串
- `sep` - `objs` 为多个值时，使用此参数作为间隔，默认为一个空格
- `end` - 末尾添加的字符，默认为一个换行
- `file` - 打印的目标，应当为具有 `write(string)` 方法的对象，默认为 `None`，表示打印到标准输出
- `flush` - 是否强制刷新，默认为 `False`

返回值

无

示例

```
print("hello world")                # 基础打印

print("A", "B", "C", "D", sep="-")  # 使用 "-" 作为分隔

print("Welcome", end="!!!\n")       # 使用 "!!!\n" 作为结尾
```

推荐阅读

- Python 官方文档
- 内置函数 input

Python 的内置函数 property

说明：创建属性。

函数说明

```
class property(getter=None, setter=None, deleter=None, doc=None):
    '''
    创建属性

    :param getter: 获取属性的方法
    :param setter: 修改属性的方法
    :param deleter: 删除属性的方法
    :param doc: 属性的说明
    :return: 属性
    '''
```

说明

通过 `getter`， `setter`， `deleter` 创建属性。通常以 装饰器 的形式使用。

不创建 `deleter` 能够让属性不可被删除；不创建 `setter` 能够让属性不可被修改。

参数

- `getter` - 获取属性的方法
- `setter` - 修改属性的方法
- `deleter` - 删除属性的方法
- `doc` - 属性的说明，默认为 `None`

返回值

封装的属性。

示例

```
class Cat:
    def __init__(self):
        self.__name = None

    # getter
    @property
    def name(self) -> str:
```

```
        print('getter')
        return self.__name

    # getter，不创建 `setter` 能够让属性不可被修改
    @name.setter
    def name(self, name):
        print('setter')
        self.__name = name

    # deleter，不创建 `deleter` 能够让属性不可被删除
    @name.deleter
    def name(self):
        print('deleter')
        del self.__name

cat = Cat()
cat.name            # 调用 getter
cat.name = 'Tom'    # 调用 setter
del cat.name        # 调用 deleter
```

推荐阅读

- Python 官方文档
- 内置函数 staticmethod
- 内置函数 classmethod

Python 的内置函数 range

说明：生成范围序列。

函数说明

```
class range(stop):
    '''
    生成 0 到 stop （不含 stop）的范围序列

    :param stop: 停止值
    :return: 范围序列
    '''
```

说明

通过停止值创建 `range` 对象。

创建的 `range` 对象从 0 开始，到停止值结束，**不包含** 停止值。

参数

- `stop` - 停止值

返回值

通过范围创建的 `range` 对象。

```
class range(start, stop, step=1):
    '''
    生成 start 到 stop （不含 stop）的范围序列

    :param start: 起始值
    :param stop: 停止值
    :param step: 步长
    :return: 范围序列
    '''
```

说明

通过起始值，停止值和步长创建 `range` 对象。

创建的 `range` 对象从起始值开始，到停止值结束，**不包含** 停止值；相邻的两个值之间相差一个步长。

注意，创建的 `range` 对象中 **不包含** 停止值。

参数

- `start` - 起始值
- `stop` - 停止值
- `step` - 步长，默认为 1

返回值

通过范围创建的 `range` 对象。

示例

```
print(range(10))
print(list(range(10)))
print(list(range(3, 10)))
print(list(range(3, 10, 3)))
```

推荐阅读

- Python 官方文档

Python 的内置函数 repr

说明：转换为字符串（调用对象的 `__repr__` 方法）。

函数说明

```
def repr(obj):
    '''
    转换为字符串（调用对象的 `__repr__` 方法）

    :param obj: 一个对象
    :return: obj.__repr__ 返回的字符串
    '''
```

说明

通过调用对象的 `__repr__` 方法，将其转换为字符串。

如果不希望出现非 ASCII 字符，可以使用 ascii 函数。

参数

- `obj` - 要转换的对象

返回值

转换后的字符串。

示例

```
class Cat:
    def __repr__(self) -> str:
        return "I am a 喵喵"

cat = Cat()

print(repr(cat))
```

推荐阅读

- Python 官方文档

Python 的内置函数 reversed

说明：将可迭代对象反转。

函数说明

```
def reversed(iterable):
    '''
    将可迭代对象反转

    :param iterable: 要反转的可迭代对象
    :return: 反转后的可迭代对象
    '''
```

说明

将可迭代对象反转。

参数的可迭代对象需要满足以下条件之一：

1. 对象具有 `__reversed__` 方法
2. 对象具有 `__len__` 方法和 `__getitem__` 方法，`__getitem__` 需要接收从 0 开始的整数参数

参数

- `iterable` - 要反转的可迭代对象

返回值

反转后的可迭代对象。

示例

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(reversed(numbers))
print(list(reversed(numbers)))
```

推荐阅读

- Python 官方文档
- 内置函数 list

- 内置函数 tuple

Python 的内置函数 round

说明：求近似值。

函数说明

```
def round(number, ndigits=None):
    '''
    求近似值

    :param number: 要计算的值
    :param ndigits: 保留的位数
    :return: 近似值
    '''
```

说明

求近似值，保留的位向最近的偶数舍入。

对于支持 `round` 的内置类型，结果会舍入至最接近的 10 的负 `ndigits` 次幂的倍数；如果与两个倍数同样接近，则优先选用偶数倍。

- `ndigits` 为 0 或 `None` 时，保留整数；10 的负 `ndigits` 次幂即 10 的 0 次幂，也就是 1；优先选用 1 的偶数倍，也就是优先选用偶数本身。
- `ndigits` 为负数时，相应的整数位优先取偶数。
- `ndigits` 为正数时，相应的小数位优先取偶数；但由于浮点数精度误差的影响，结果可能不符合预期。

高精度需求的场景建议使用 `decimal` 模块。

参数

- `number` - 要求近似值的值
- `ndigits` - 结果要保留的小数位数，可用为负值，表示保留到整数位，默认为 `None`，表示保留整数

返回值

近似值。

示例

```python
print(round(3.1415926))          # 保留整数
print(round(3.1415926, 2))       # 保留两位小数

print(round(2.71828))            # 保留整数
print(round(2.71828, 2))         # 保留两位小数

print(round(2233.71828, -2))     # 保留到百位

print(round(2.5))                # 输出：2（优先选用偶数）
print(round(3.5))                # 输出：4（优先选用偶数）

print(round(25, -1))             # 输出：20（十位优先选用偶数）
print(round(35, -1))             # 输出：40（十位优先选用偶数）

print(round(0.25, 1))            # 输出：0.2（优先选用 0.1 的偶数倍）
print(round(0.35, 1))            # 输出：0.3（浮点数精度误差）
```

推荐阅读

- Python 官方文档

Python 的内置函数 set

说明：类型转换为 `set`。

函数说明

```
class set():
    '''
    创建一个空集合

    :return: 创建出的空集合
    '''
```

说明

创建一个空集合

参数

无

返回值

创建出的空集合。

```
class set(iterable):
    '''
    类型转换为 set

    :param iterable: 一个可迭代对象
    :return: 创建出的集合
    '''
```

说明

通过可迭代对象创建集合。

集合元素不可重复，因此参数对象中的重复值会被过滤为一个。

参数

- `iterable` - 一个可迭代对象

返回值

创建出的集合。

示例

```
print(set()) # 空集合，不能使用 {}，因为 {} 创建的是空字典

print(set([1, 1, 1, 2, 2, 3, 3, 4, 5, 5, 6]))
```

推荐阅读

- Python 官方文档
- 内置函数 frozenset
- 内置函数 dict
- 内置函数 list
- 内置函数 tuple

Python 的内置函数 setattr

说明：设置属性的值。

函数说明

```
def setattr(obj, name:str, value):
    '''
    设置属性的值

    :param obj: 一个对象
    :param name: 属性的名字
    :param value: 属性的值
    '''
```

说明

设置对象属性的值；如果属性不存在则会添加属性。

> 请参考 hasattr、delattr 和 getattr 等函数。

参数

- `obj` - 要设置属性的对象
- `name` - 要设置的属性名
- `value` - 要设置的值

返回值

无

示例

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p = Person("Alice", 30)

setattr(p, 'name', 'Doro')  # 修改属性
setattr(p, 'age', 15)       # 修改属性
setattr(p, 'pet', 'dog')    # 添加属性
```

```
print(p.name)
print(p.age)
print(p.pet)
```

推荐阅读

- Python 官方文档
- 内置函数 getattr
- 内置函数 hasattr
- 内置函数 delattr

Python 的内置函数 slice

说明：生成切片。

函数说明

```
class slice(stop):
    '''
    生成 0 到 stop （不含 stop）的切片

    :param stop: 停止值
    :return: 切片
    '''
```

说明

通过停止值创建 `slice` 对象。

切片对象从 0 开始，到停止值结束，**不包含** 停止值。

参数

- `stop` - 停止值

返回值

通过范围创建的 `slice` 对象。

```
class slice(start, stop, step=1):
    '''
    生成 start 到 stop （不含 stop）的切片

    :param start: 起始值
    :param stop: 停止值
    :param step: 步长
    :return: 切片
    '''
```

说明

通过起始值，停止值和步长创建 `slice` 对象。

创建的 `slice` 对象从起始值开始，到停止值结束，**不包含** 停止值；相邻的两个值之间相差一个步长。

注意，创建的 `slice` 对象中 **不包含** 停止值。

参数

- `start` - 起始值
- `stop` - 停止值
- `step` - 步长，默认为 1

返回值

通过范围创建的 `slice` 对象。

示例

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

print(numbers[slice(3)])                # 0 到 3，不含 3
print(numbers[slice(3, 10)])            # 3 到 10，不含 10
print(numbers[slice(3, 10, 3)])         # 3 到 10，不含 10，步长为 3

print(numbers[slice(-3)])               # 0 到 -3（倒数第三个元素），不含 -3
print(numbers[slice(-3, -7, -1)])       # -3（倒数第三个元素）到 -7（倒数第七个元素） 到，不含 -7
```

推荐阅读

- Python 官方文档
- 内置函数 list
- 内置函数 tuple
- 内置函数 str

Python 的内置函数 sorted

说明：排序。

函数说明

```
def sorted(iterable, /, *, key=None, reverse=False):
    '''
    排序

    :param iterable: 一个可迭代对象
    :param key: 回调函数，返回用于比较的字段
    :param reverse: 是否反向
    :return: 排序后的列表
    '''
```

说明

将一个可迭代对象内的元素按照从升序（小到大的顺序）进行排序，得到一个排序后的列表。

如果要在一个列表内进行原地排序，可以使用 `list.sort` 方法。

参数

- `iterable` - 要排序的可迭代对象
- `key` - 回调函数，可迭代对象的元素会作为参数传给此函数，通过此函数的返回值进行比较，默认为 `None`，即直接比较原本的值
- `reverse` - 是否反转，即按照降序（从大到小的顺序）进行排序，默认为 `False`

返回值

排序后的列表；无论入参是何种可迭代对象，返回值类型均为 `list`。

示例

```
numbers:list[int] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(sorted(numbers))
print(sorted(numbers, reverse=True))

pets = [('Tom', 8), ('Jerry', 9), ('Spike', 5), ('Tuffy', 3)]
print(sorted(pets))
print(sorted(pets, key=lambda x:x[1]))
```

推荐阅读

- Python 官方文档
- 内置函数 list
- 内置函数 tuple

Python 的内置函数 staticmethod

说明：把一个方法封装成静态方法。

函数说明

```
def staticmethod(fn):
    '''
    把一个方法封装成静态方法

    :param fn: 要封装的方法
    :return: 封装后的方法
    '''
```

说明

将一个方法封装成静态方法，通常以 装饰器 的形式使用。

静态方法没有隐含参数。要声明一个静态方法，按惯例请使用以下方案：

```
class C:
    @staticmethod
    def fn(arg1, arg2):
        pass
```

静态方法的调用可以在类上进行，也可以在实例对象上进行，不会传递隐含参数。

参考 classmethod 函数。

参数

- `fn` - 要封装成静态方法的函数

返回值

封装成静态方法后的函数。

示例

```
class Cat:
    name = '薛定谔'

    @staticmethod
```

```
    def speak():
        print(f'喵喵喵，我叫{Cat.name}')    # 静态方法没有隐含参数，因此需要通过类名访问类的属性

# 通过类调用
Cat.speak()

# 通过对象调用
cat = Cat()
cat.speak()
```

推荐阅读

- Python 官方文档
- 内置函数 classmethod
- 内置函数 property

Python 的内置函数 str

说明：类型转换为 `str`。

函数说明

```
class str(x=''):
    '''
    类型转换为 str

    :param x: 一个变量
    :return: 转换为 str 后的值
    '''
```

说明

通过调用对象的 `__str__` 方法，将一个变量转换为字符串。

如果对象没有 `__str__` 方法，则调用 `__repr__` 方法。

参数

- `x` - 要转换的变量，默认为一个空字符串

返回值

转换为 `str` 后的值。

```
class str(x:bytes=b'', encoding='utf-8', errors='strict'):
    '''
    将一个 bytes 转换为字符串

    :param x: 一个 bytes 类型的变量
    :param encoding: 编码，默认为 UTF-8
    :param errors: 指定如何处理编解码错误
    :return: 转换为 str 后的值
    '''
```

说明

将一个 bytes 转换为字符串。

参数

- `x` - 一个 bytes 类型的变量
- `encoding` - 编码，默认为 UTF-8
- `errors` 指定如何处理编解码错误，默认为 `'strict'`

返回值

转换为 `str` 后的值。

编码错误的处理方式

- `None` 或 `'strict'` - 编码错误时产生 `ValueError` 异常
- `'ignore'` - 忽略错误，通常会丢失数据
- `’replace'` - 使用 `'?'` 替换有错误的位置
- `'surrogateescape'` - 使用 `U+DC08` 至 `U+DCFF` 之间的值表示不正确的字节
- `'xmlcharrefreplace'` - 编码格式不支持的字符将被替换为相应的 XML 字符引用，仅在写入时可用
- `'backslashreplace'` - 使用反斜杠转义表示错误的数据
- `'namereplace'` - 使用 `\N{...}` 转义表示错误的数据

示例

```
# print 内部本就将对象转换为字符串打印，因此看起来没有变化
print(str(2e10))
print(str([1,2,3,4,5]))
print(str((1,)))

# 编码
print(str(b'Python', encoding='utf-8'))

# 自定义
class Person:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"Person: {self.name}"

p = Person("Alice")
print(str(p))
```

推荐阅读

- Python 官方文档

- 内置函数 repr
- 内置函数 slice
- 内置函数 list
- 内置函数 tuple

Python 的内置函数 sum

说明：求和。

函数说明

```
def sum(iterable, /, start=0):
    '''
    求和

    :param iterable: 一个可迭代对象
    :param start: 起始值
    :return: 和
    '''
```

说明

对一个可迭代对象内的元素求和。

`sum` 可以处理各种数值类型，包括：

- 整数（`int`）
- 浮点数（float）
- 复数（complex）
- 其他实现了 `__add__` 方法的对象

参数

- `iterable` - 要进行求和的可迭代对象
- `start` - 和的初始值

返回值

求得的和。

示例

```
print(sum([1,2,3,4,5,6,7,8,9]))          # 列表求和
print(sum([1,2,3,4,5,6,7,8,9], 1000))    # 带初始值的列表求和
```

推荐阅读

- Python 官方文档

Python 的内置函数 super

说明：获取父类的代理对象。

函数说明

```
class super():
    '''
    获取父类的代理对象

    :return: 当前类的父类的代理对象
    '''
```

说明

获取父类的代理对象。

参数

无

返回值

当前类的父类代理对象。

```
class super(type, obj=None):
    '''
    获取父类的代理对象

    :param type: 要获取父类的类型
    :param obj: 要获取父类的对象
    :return: 获取父类的代理对象
    '''
```

说明

获取父类的代理对象。

参数

- `type` - 要获取父类的类型
- `obj` - 要获取父类的对象，默认为 `None`

返回值

父类的代理对象。

示例

```
class Pet:
    def speak(self):
        print('某种动物在说话')

class Cat(Pet):
    def speak(self):
        print('喵喵喵')

    def super_speak(self):
        super().speak()

cat = Cat()
cat.speak()
cat.super_speak()
super(Cat, cat).speak()
```

推荐阅读

- Python 官方文档
- 继承和多态
- 内置函数 issubclass
- 内置函数 isinstance

Python 的内置函数 tuple

说明：类型转换为 `tuple`。

函数说明

```
class tuple():
    '''
    创建一个空元组

    :return: 空元组
    '''
```

说明

创建一个空元组

参数

无

返回值

空元组

```
class tuple(iterable):
    '''
    类型转换为 tuple

    :param iterable: 一个可迭代对象
    :return: 转换为 tuple 后的值
    '''
```

说明

将变量转换为 `tuple` 类型

参数

- `iterable` - 要转换的可迭代对象

返回值

转换为 `tuple` 后的元组

示例

```
print(tuple([1, 1, 1, 2, 2, 3, 3, 4, 5, 5, 6]))
```

推荐阅读

- Python 官方文档
- 内置函数 list
- 内置函数 dict
- 内置函数 set
- 内置函数 frozenset
- 内置函数 slice

Python 的内置函数 type

函数说明

```
class type(obj):
    '''
    获取类型的代理对象

    :param obj: 一个变量
    :return: obj 的类型
    '''
```

说明

获取类型的代理对象。

参数

- `obj` - 要获取类型的对象

返回值

`obj` 的类型

```
class type(name, bases, dict, **kwds):
    '''
    创建类型

    :param name: 类型名
    :param bases: 基类
    :param dict: 属性列表
    :param kwds: 属性列表
    :return: 新的类型
    '''
```

说明

创建类型，相当于 `class`。

参数

- `name` - 类型的名称

- `base` - 要继承的基类
- `dict` - 字典表示的属性列表
- `kwds` - 任意关键字参数的属性类型

返回值

创建的类型。

示例

```
print(type(233))
print(type(3.14))
print(type('hello world'))
print(type([1, 1, 1, 2, 2, 3, 3, 4, 5, 5, 6]))

obj = type('MyClass', (), {'name':'Alice', 'age':18})
print(obj, obj.name, obj.age)
```

推荐阅读

- Python 官方文档
- 内置函数 isinstance
- 内置函数 isinstance

Python 的内置函数 vars

说明：获取对象属性的字典。

函数说明

```
def vars():
    '''
    获取对象属性的字典

    :return: 当前对象属性的字典
    '''
```

说明

通过读取当前对象的 `__dict__` 属性，获取当前对象属性的字典。

参数

无

返回值

当前对象属性的字典

```
def vars(obj):
    '''
    获取对象属性的字典

    :param obj: 一个对象
    :return: obj 属性的字典
    '''
```

说明

通过读取对象的 `__dict__` 属性，获取对象属性的字典。

参数

- `obj` - 要读取属性字典的对象

返回值

对象的属性字典

示例

```
with open(__file__) as fp:
    print('vars 只包含属性：', vars(fp))
    print('dir 包含属性和方法的名称：', dir(fp))
```

推荐阅读

- Python 官方文档
- 内置函数 dir
- 内置函数 globals
- 内置函数 locals

Python 的内置函数 zip

说明：将多个可迭代对象组合成元素为元组的可迭代对象。

```
def zip(*iterables, strict=False):
    '''
    将多个可迭代对象组合成元素为元组的可迭代对

    :param iterables: 任意个可迭代对象
    :param strict: 是否采用严格模式
    :return: 合成的元组可迭代对象
    '''
```

说明

将多个可迭代对象组合成元素为元组的可迭代对。

- 严格模式下各个可迭代对象长度必须相等，否则会产生 `ValueError` 异常
- 非严格模式下，按照最短的参数长度进行合成

参数

- `iterables` - 任意个可迭代对象
- `strict` - 是否采用严格模式，默认为 `False`，即不采用严格模式

返回值

合成的元组可迭代对象

示例

```
names = ['Tom', 'Jerry', 'Spike', 'Tuffy']
scores = [88, 99, 66, 33]
ages = [8, 7, 10]

print(zip(names, scores))
print(list(zip(names, scores)))

print(list(zip(names, ages)))

try:
    print(list(zip(names, ages, strict=True)))
```

```
except:
    print("严格模式下失败")
```

推荐阅读

- Python 官方文档
- 内置函数 list
- 内置函数 tuple