

内置常量

有少数的常量存在于内置命名空间中。 它们是：

`False`

`bool` 类型的假值。 给 `False` 赋值是非法的并会引发 [SyntaxError](#)。

`True`

`bool` 类型的真值。 给 `True` 赋值是非法的并会引发 [SyntaxError](#)。

`None`

通常被用来代表空值的对象，例如未向某个函数传入默认参数时。 向 `None` 赋值是非法的并会引发 [SyntaxError](#)。`None` 是 [NoneType](#) 类型的唯一实例。

`NotImplemented`

一个应当由双目运算特殊方法（如 `__eq__()`, `__lt__()`, `__add__()`, `__rsub__()` 等）返回的特殊值，用来表明该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。 它不应在布尔上下文中被求值。`NotImplemented` 是 [types.NotImplementedType](#) 类型的唯一实例。

备注: 当一个双目（或原地）方法返回 `NotImplemented` 时解释器将尝试对另一种类型（或其他回退操作，具体取决于所用的运算符）的反射操作。 如果所有尝试都返回 `NotImplemented`，解释器将引发适当的异常。 错误地返回 `NotImplemented` 将导致误导性的错误消息或 `NotImplemented` 值被返回给 Python 代码。

参见 [实现算术运算](#) 为例。

小心: `NotImplemented` 和 `NotImplementedError` 不能互相替代。 此常量应当仅以上文所描述的方式使用；请参阅 [NotImplementedError](#) 了解正确使用该异常的相关细节。

在 3.9 版本发生变更: 在布尔运算上下文中对 `NotImplemented` 求值的做法已被弃用。

在 3.14 版本发生变更: 在布尔运算上下文中对 `NotImplemented` 求值现在会引发 `TypeError`。 在之前版本中它会被求值为 `True` 并将自 Python 3.9 起发出 [DeprecationWarning](#)。

`Ellipsis`

与省略号字面值 "`...`" 相同，这是一个常用于表示省略内容的对象。 可以对 `Ellipsis` 进行赋值，但对 `...` 进行赋值会引发 [SyntaxError](#)。`Ellipsis` 是 [types.EllipsisType](#) 类型的唯一实例。

`__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。

备注: 变量名 `None`, `False`, `True` 和 `__debug__` 无法重新赋值（赋值给它们，即使是属性名，将引发 [SyntaxError](#)），所以它们可以被认为是“真正的”常数。

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(code=None)`
`exit(code=None)`

当对象被打印时，会打印一条消息如 "Use quit() or Ctrl-D (i.e. EOF) to exit"，当在交互解释器中直接访问或作为函数调用时，将引发 [SystemExit](#) 并附带指定的退出码。

`help`

该对象在打印时会输出提示信息："输入 help() 可获取交互式帮助，或输入 help(object) 查看特定对象的帮助信息"；当在交互式解释器中直接访问该对象时，则会调用内置的帮助系统（参见 [help\(\)](#) 函数）。

`copyright`
`credits`

打印或调用的对象分别打印版权或作者的文本。

`license`

当打印此对象时，会打印出一条消息"Type license() to see the full license text"，当调用此对象时，将以分页形式显示完整的许可证文本（每次显示一屏）。